# *DC2-PC*

Motion Control Card

**User's Manual**

Revision 1.3a

**LIMITED WARRANTY**

All products manufactured by PRECISION MICRO CONTROL CORPORATION are guaranteed to be free from defects in material and workmanship, for a period of five years after date of shipment. Liability is limited to FOB Factory repair, or replacement, of the product. Other products supplied as part of the system carry the warranty of the manufacturer.

PRECISION MICRO CONTROL CORPORATION does not assume any liability for improper use or installation or consequential damage.



Information in this document is subject to change without notice.

IBM and IBM-PC are registered trademarks of International Business Machines Corporation.

Microsoft, MS-DOS, QuickC and QuickBasic are registered trademarks of Microsoft Corporation.

# TABLE OF CONTENTS

## List of figures

# 1.0 INTRODUCTION

This document describes the installation and operation of the **DC2-PC** motion control card.

While there are several variations of the **DC2-PC** product, with model numbers varying in the last two digits (e.g. **DC2-PC110**), this manual serves for all models. Henceforth in this manual, the term "**DC2**" will be used to refer to all models of the product.

The **DC2** is a dedicated 2-axes servo controller. With its on board micro-controller and firmware, the **DC2** is capable of accurately controlling the position or velocity of two independent servos. The **DC2** interfaces with incremental encoders on its inputs for monitoring the servo's position, while it controls the torque or velocity of the motor with its outputs.

The **DC2** also provides 2 axes of programmable stepper motor indexing with 16 bit position and velocity resolution.

The computer logic in the **DC2** system is programmed to interpret and execute commands sent to it by the user. Commands are provided for setting parameters related to controlling the servos. A typical command sequence will set a servo's velocity, acceleration and deceleration, then cause it to move to a specific position. The ability to remember sequences of commands, called "macro commands", provides the user with a powerful tool for implementing various control requirements.

A battery and power-fail detection circuitry located on the **DC2** provides reliable backup of the on-board memory. This, along with the **DC2**'s ability to execute a specific macro on power-up, allows the **DC2** to function totally on its own in certain applications.

# 1.1 SPECIFICATIONS

| | |
|---|---|
| Function: | 2 Axes Servo Controller + 2 Axes Stepper Motor Indexer |
| Installation: | IBM-PC/XT/AT compatible computer or stand-alone |
| Motor Outputs: | |
| DC2-PC100 | Analog Signal (+/- 10 vdc @ 10 ma, 12 bit) |
| DC2-PC110 | PWM (30KHz) Motor Drive (1amp @ 12 to 24 vdc, 8 bit) |
| DC2-PC140 10 | Jumper selectable for each axis either Analog Signal (+/- vdc @ 10 ma, 12 bit) or PWM (30 KHz) Motor Drive (1 amp @ 12 to 24 vdc, 8 bit) |
| Operating Modes: | Position, Velocity, Torque and Gain |
| Filter Algorithm: | PID with Feed Forward |
| Filter Update Rate: | 1.0 KHz |
| Trajectory Generator: | Trapezoidal with Independent Acceleration and Deceleration |
| Position and Velocity Res.: | Servo 32 bit, Stepper (indexers) 16 bit |
| Servo Position Feedback: | Incremental Encoder with Index |
| Encoder and Index Input: | Differential or single ended |
| Encoder Supply Voltage: | 5 or 12 vdc, jumper selectable |
| Encoder & Index Input Voltage: | -7 to +7 vdc max. |
| Max. Encoder Count Rate: | 1,000,000 Quadrature Counts / Sec. |
| Dedicated Inputs: | Limit+, Limit-, Coarse Home (TTL or Opto Iso.) |
| Dedicated Outputs: | Amplifier Enable (TTL level) |
| User definable Digital I/O: | 16 TTL level (0 - 5 vdc) (max sink/source 1ma) |
| A-D input Channels: | 4, 8 bit resolution (0 - 5 vdc) |
| Communication Interfaces: | IBM-PC/XT/AT or Compatible Bus (8 bit) or RS-232 Serial (network support optional) |
| PC Bus Interface: | 4 Registers (Data In, Data Out, Status and Attention) in I/O (default) or Memory Address Space |
| RS-232 Serial Interface: | 300 - 19,200 Baud, Hardware Handshake or XON-XOFF Handshake Protocol |
| Power Supply req., logic: | +5V @ .6a, +12V @.1a, -12V @ .05a |
| Power Supply req., motor: | 12 to 24 vdc @ (current to match motor requirements) |
| Battery: CR2025 | 3 vdc (2.2 vdc minimum) lithium, 140 mah (10 ave. life) |
| Form Factor: | Full Size PC card (4.3" x 13.6") |
| Weight: | 9 oz |

## 1.2 OVERVIEW

In order to make the **DC2** as versatile as possible, many features and capabilities have been built into it. This manual has been divided into sections and appendices that guide the reader through the task of becoming familiar with the **DC2**. The manual will also serve as a detailed reference.

The remainder of section 1 of this manual details the theory of operation.

Section 2 describes initial controller setup, including connector wiring and jumper settings.

Section 3 describes the available communication interfaces.

Section 4 is a tutorial on controller operation.

Section 5 describes the control of servo motors.

Section 6 covers the stepper indexing capability of the board.

Section 7 describes specific servo motion functions.

Section 8 describes the 'undedicated' I/O.

Section 9 describes each of the **DC2** commands in detail.

Appendix A is a summary of the **DC2** command codes (used by the binary interface).

Appendix B lists the pinouts of the **DC2**'s connectors and describes the jumper functions.

Appendix C describes the binary mode command interface.

Appendices D and E contain the listings of two example programs ("C" and Basic) for communicating to the **DC2**, when it is installed in an IBM-PC/XT/AT.

Appendix F contains tables related to the **DC2**'s operation.

And appendix G contains a trouble shooting guide.

# 1.3 THEORY OF OPERATION

The servo control functions of the **DC2** are performed by a single micro controller located on the board. This device executes instructions stored in memory devices on the board. These instructions, called "firmware", cause the micro controller to monitor inputs and activate outputs in order to control the attached servos.

The **DC2** has the ability to interpret commands sent to it by a host computer or stored in its on-board memory. These commands can implement functions from simple servo motions to complex sequencing operations.

### The DC2 Micro controller

The micro controller splits its time servicing the command interpreter and the two servo axes. To achieve precise velocity and position control, the **DC2** performs a series of calculations for both axes once per millisecond. These calculations implement the trajectory generators which determine the ideal positions of the servos, and PID loops for driving the servos to those positions.

The trajectory generator calculates a "motion profile" using the specified maximum velocity, acceleration, deceleration and target destination. The motion profile dictates the ideal position of an axis for each one millisecond interval.

Each servo axis utilizes a velocity feed-forward and a position feedback loop to control the motor. For applications requiring an external amplifier, the **DC2-PC100** (and **DC2-PC140)** provides a +/- 10 volt analog control signal output to an external servo amplifier. For direct motor drive applications, the **DC2-PC110** (and **DC2-PC140)** outputs a 30 KHz PWM signal capable of driving as much as 1 amp of current.

### Position Feedback

An incremental encoder input provides feedback information for closing the position loop. Every millisecond the micro controller subtracts the actual position (feedback position) from the desired position (trajectory generator position) for both axes, and the resulting position error is processed by the digital filter. The output of the digital filter and the velocity feed-forward are combined to set the motor control signal output. The motor then moves toward its desired position.

The micro controller monitors the motor's position via an incremental encoder. The two quadrature signals A+ and B+ (A+, A-, B+, and B- if using a differential encoder) from the encoder are used to keep track of the absolute position of the motor. Each time a logic transition occurs at one of the quadrature inputs, the servo controller position counter is incremented or decremented accordingly. This provides four times the resolution over the number of lines provided by the encoder. The encoder inputs are buffered by a differential line receiver on the module. Configuration jumpers on the **DC2** allow this receiver to be configured for single ended input signals.

### PID Filter (servo loop)

The **DC2** utilizes a digital "Proportional Integral Derivative" (PID) with Feedforward filter to

control the response characteristics of the servo. The PID loop, which executes every millisecond, calculates the difference between a servo's current position and the "desired position" (determined by the trajectory generator). This difference is then multiplied by a gain constant and applied to the **DC2** output as a restoring force. The PID loop also performs other adjustments to the output value in order to stabilize the servo and reduce positioning errors.

The PID loop algorithm is as follows:

$$U(t) = Kp*E(t) - Kd*[E(t)-E(t-1)] + Ki*\text{sum } E(t) + Kv*V(t)$$

where
U(t) = Servo Drive Output at time t where:
DC2-PC100 output units = 0.00488 volts/count
DC2-PC110 output units = 0.39 percent/count
E(t) = Position error (difference between desired and current position) at time t
V(t) = Velocity at time t
Kp = Proportional Gain where:
DC2-PC100 proportional gain units = 0.0000190 volts/error
DC2-PC110 proportional gain units = 0.00152 percent/error
Kd = Derivative Gain where:
DC2-PC100 derivative gain units = 0.0000000190 volts/(error/second)
DC2-PC110 derivative gain units = 0.00000152 percent/(error/second)
Ki = Integral Gain where:
DC2-PC100 integral gain units = 0.0190 volts/(error*second)
DC2-PC110 integral gain units = 0.152 percent/(error*second)
Kv = Velocity Gain where:
DC2-PC100 velocity gain units = 0.0000000190 volts/(encoder counts/second)
DC2-PC110 velocity gain units = 0.00000152 percent/(encoder counts/second)

The first term, the proportional term, adjusts the amount of restoring force (stiffness) applied to the servo for a given amount of position error. The larger the error, the greater restoring force. While increasing the proportional gain will reduce position error, it can also cause the servo to oscillate.

The second term, the derivative term, provides dampening as a force proportional to the rate of change of position error. It adjusts the restoring force based on the change of position error in time. The larger the derivative gain, the greater the dampening effect. The sampling interval associated with the derivative term is user-selectable; this capability enables the servo controller to control a wider range of inertial loads.

The third term, the integration term, provides a restoring force that grows with time. In order to achieve zero position errors in systems that have "sticktion", the integral gain can accumulate the error, increasing the restoring force until the error is corrected. The integral gain sets the rate that the error accumulates.

The forth term, the velocity gain (also known as "Feed-Forward") can be used to reduce the position error during motion. This term uses velocity gain and the desired velocity to adjust the servo drive output.

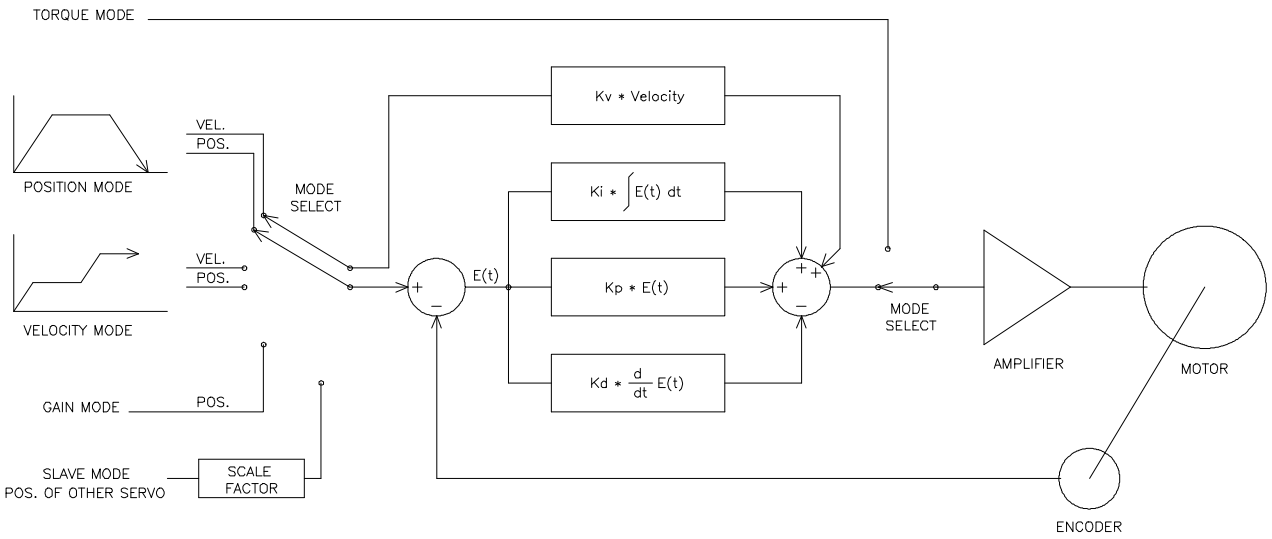


**Figure 1: DC2 servo motor control (PID) architecture**

**Tuning the servo (PID parameters)**

Before attempting to implement servo motion an axis must be "tuned". Servo tuning is the setting of the PID (Proportional, Integral, and Derivative) gain parameters to match the requirements (desired servo response, mechanical load, stiction, etc...) of the application. The gain parameters are set by issuing commands to the **DC2** during initialization, and will normally be left at those values during servo operation. The Windows based DC2 Tuning Utility (pictured on the next page) can be used to quantify the effect of each of the PID parameters for each axis and save the parameter values to an initialization file. For non Windows environments, section 5.0 SERVO OPERATION describes command line servo tuning.

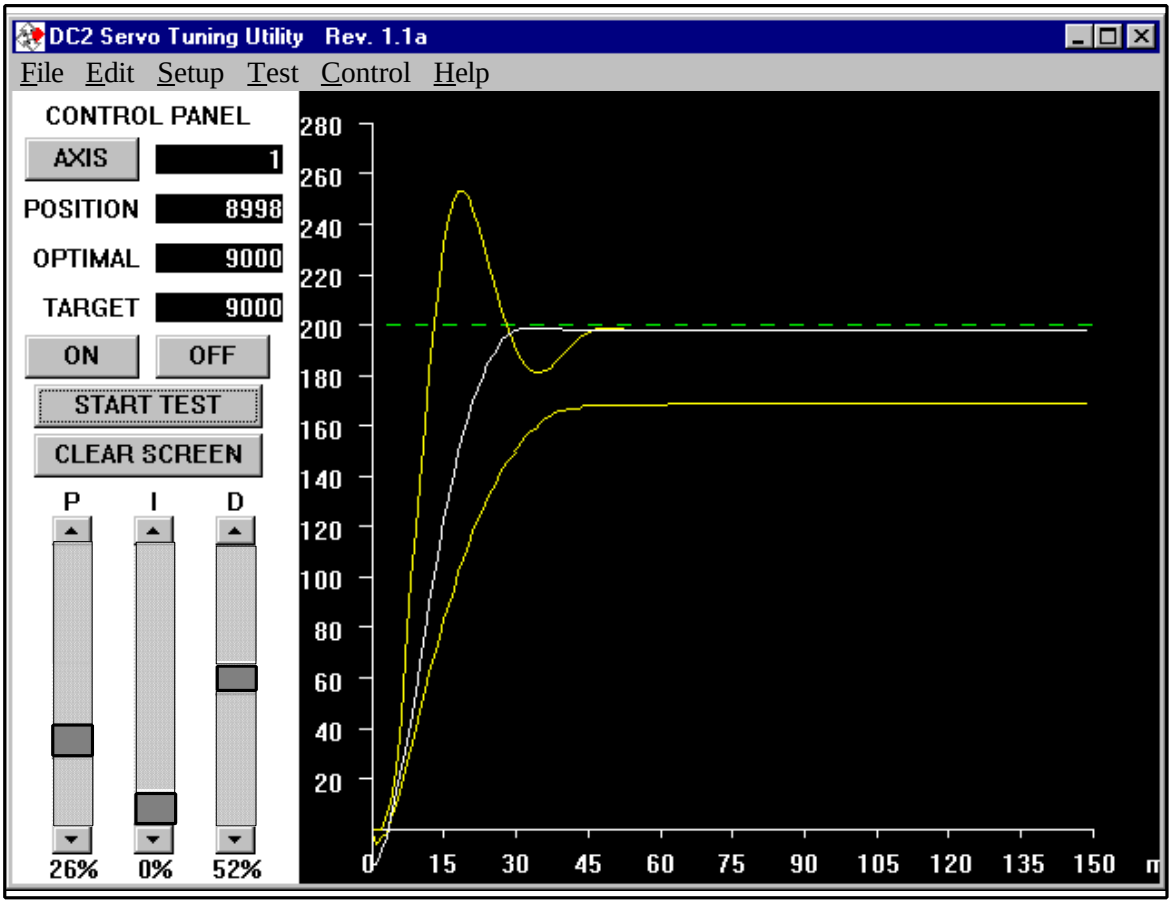


Figure 2: Windows based DC2 Tuning Utility

**Velocity Profile (Trajectory Generation)**

The trapezoidal velocity profile generator computes the desired position of the motor versus time. In the position mode of operation, the host processor specifies acceleration, maximum velocity, and position. The **DC2** uses this information to affect the move by accelerating as specified until the maximum velocity is reached or until deceleration must begin to stop at the specified final position. At any time during the move the maximum velocity and/or the target position may be changed “on-the-fly”, and the motor will accelerate or decelerate accordingly.

When operating in the velocity mode, the motor accelerates to the specified velocity at the specified acceleration rate and maintains the specified velocity until commanded to stop. The velocity is maintained by advancing the desired position at a constant rate. If there are disturbances to the motion during velocity mode operation, the long-time average velocity remains constant. If the motor is unable to maintain the specified velocity (which could be caused by a mechanical bind), the desired position will continue to be increased, resulting

in a very large position error. If this condition goes undetected, and the impeding force on the motor is subsequently released, the motor could reach a very high velocity in order to catch up to the desired position (which is still advancing as specified).

All trajectory parameters are 32-bit values. Position is a signed quantity. Acceleration and velocity are specified as 16-bit, positive-only integers having 16-bit fractions. The integer portion of velocity specifies how many counts per sampling interval the motor will traverse. The fractional portion designates an additional fractional count per sampling interval. Although the position resolution of the servo controller is limited to integer counts, the fractional counts provide increased average velocity resolution. Acceleration and deceleration are treated in the same manner. Each sampling interval the commanded acceleration value is added to the current desired velocity to generate a new desired velocity (unless the command velocity has been reached).

**Modes of Motion**

In order to perform various motion functions, the **DC2** supports five modes of operation:

In "Position" mode, the trajectory generator calculates a motion profile to cause the servo to move to a specified position, known as the target. The motion is constrained to predefined acceleration, maximum velocity and deceleration values. The **DC2** is responsible for causing the servo to move to the target in the minimum amount of time within these constraints. Because the acceleration and deceleration values are constant, this type of motion is known as a "Trapezoidal" velocity profile.

In "Velocity" mode no target position is given. The trajectory generator causes the servo to accelerate to the specified maximum velocity and move until commanded to stop.

In both the position and velocity modes, the PID loop is driving the servo to the position calculated by the trajectory generator every millisecond.

In "Gain" mode the trajectory generator is disabled and the PID loop drives the servo to the specified target position with no motion profile.

Another mode that disables the trajectory generator is "Master/Slave" mode. In this mode, the current position of the other servo is used as the desired position. The PID loop will cause the slave servo to follow the position of the master. The master servo can be in any mode and operated in its normal fashion. For "electronic gearing" applications, a scaling factor can be applied to the slave servo position to create ratios less than 1 to 1.

In "Torque" mode both the trajectory generator and PID loops are turned off. The servo drive outputs are set to constant levels specified by **DC2** commands.

## 2.0 INITIAL CONTROLLER SET-UP

The **DC2** may be installed in any available backplane connector of an IBM-PC,XT,AT compatible, or it may be used stand-alone configuration. Please note that the **DC2** does not support communication with more than one interface at a time.

When installed in an IBM-PC/XT/AT or compatible computer, power, communications, and reset control (JP1 pins 2 and 3) will be supplied through the edge connector of the **DC2**.

When used in the stand-alone configuration, a mating edge connector is required to terminate address and data line and supply power to the **DC2.** A terminated edge connector is available from Precision MicroControl (PMC P/N 70.106.A) or the connector can be fabricated by the user (see the note in appendix B page 4 for details). Connect pins 1 and 2 of jumper JP1 to use the Power OK reset circuit

The J4 connector supports the RS-232 serial communications interface. The **DC2** processes ASCII format DC2 commands from a host processor. If the host processor issues a **DC2** command that generates a response, the board will issue a response to the RS-232 interface port.

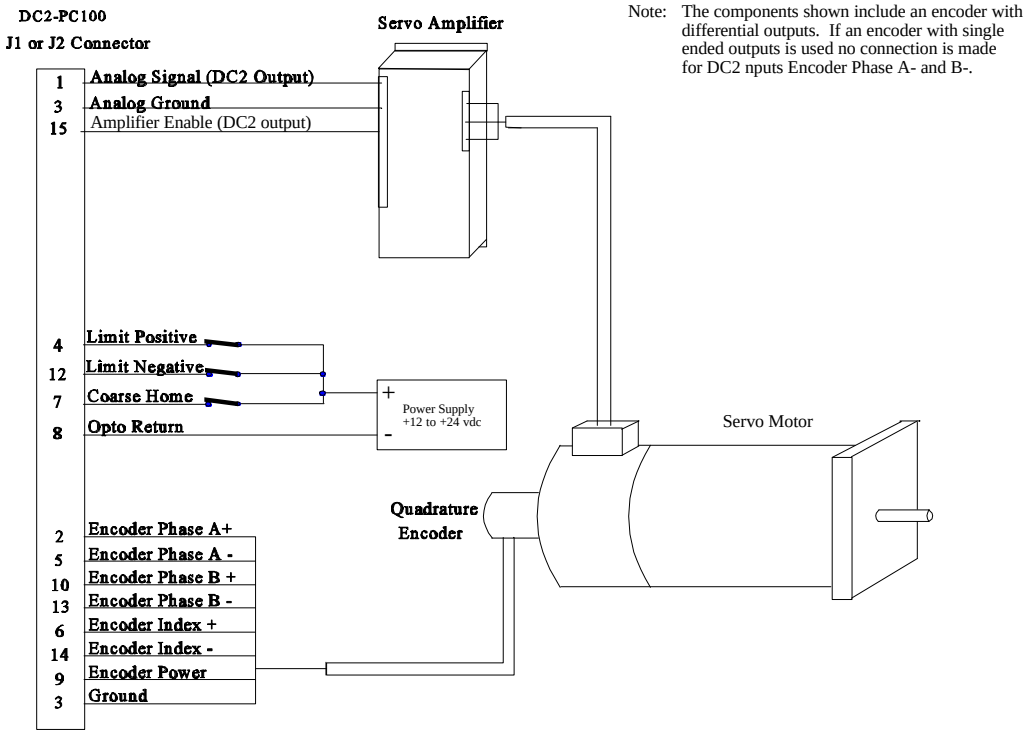


Figure 3: DC2-PC100 typical servo system interconnect

## 2.1 MOTOR CONNECTIONS

**Please refer to the figure on the next page.**

Servo motor connections to the **DC2** are made via two 15 pin D-subminiature connectors. The 15 pin D-sub's are labeled J1 and J2, and are located on the right short edge of the board. All connections to J1 are for servo axis 1, and to J2 are for servo axis 2.

In setting up the **DC2** for servo motor control, there are two options for each axis. The motor can be driven with the on board power circuits, or a control signal from the **DC2** can be connected to an external amplifier. If the motor drive requirement is 1.0 amps or less, at any voltage between 12 and 24 volts, the on board PWM amplifier may be suitable. For higher current or voltage drive requirements, an external amplifier must be used. The output of the on board Digital to Analog Converter (DAC) is variable from -10 to +10 vdc and is compatible with most amplifier inputs. Two potentiometers (POT1 & POT2) are provided to adjust the offset of the DAC outputs. POT1 is used to adjust the offset for axis one, POT2 is used to adjust the offset for axis two.

Another criteria in selecting which output to use, is the required resolution. The PWM drive has 8 bits of resolution (1 in 256) while the DAC output has 12 bits (1 in 4096). Applications requiring tighter positioning tolerances may need the higher resolution of the DAC output.

For applications with different sized servos, it is acceptable to use the PWM drive for one axis and the DAC output on the other.

Use the jumper blocks on the board to connect pins on the various jumpers mentioned in the following paragraphs:

To use the the on board PWM amplifier for axis 1, connect pins 1 and 2 of jumper JP2, and connect the motor leads to pins 1 and 11 of connector J1. To use the host PC's +12 VDC power supply to run the motor, connect pins 2 and 3 of jumper JP6. To use an external supply, connect pins 1 and 2 of jumper JP6. The external supply ground terminal should be connected to pin 3 of J5, the positive terminal should then be connected to pin 1 of J5.

Alternatively, to use an external amplifier to control axis 1, connect pins 2 and 3 of jumper JP2. Connect pin 1 of connector J1 to the amplifier signal input, and pin 3 of J1 to the amplifier signal return. Leave pin 11 of J1 unconnected and all pins on jumper JP6 unconnected.

To use the the on board PWM amplifier for axis 2, connect pins 1 and 2 of jumper JP4, connect the motor leads to pins 1 and 11 of connector J2. To use the host PC's +12 VDC power supply to run the motor, connect pins 2 and 3 of jumper JP7. To use an external supply, connect pins 1 and 2 of jumper JP7. The external supply ground terminal should be connected to pin 3 of J5, the positive terminal should then be connected to pin 2 of J5.

Alternatively, to use an external amplifier to control axis 2, connect pins 2 and 3 of jumper JP4. Connect pin 1 of connector J2 to the amplifier signal input, and pin 3 of J2 to the amplifier signal return. Leave pin 11 of J2 unconnected and all pins on jumper JP7 unconnected.

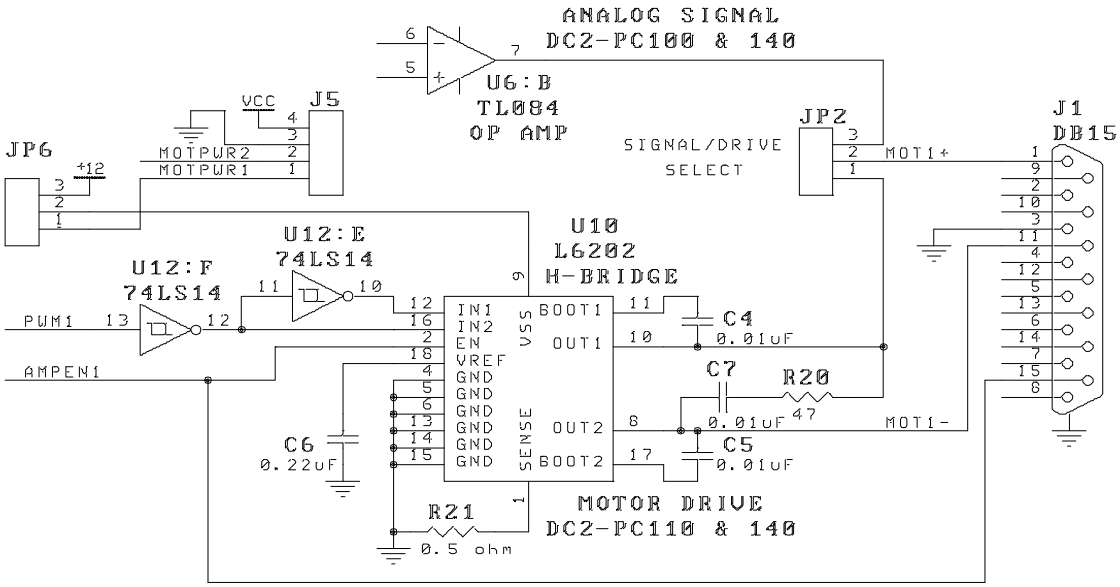


Figure 4: DC2 motor circuit (typical)

## 2.2 ENCODER CONNECTIONS

**Please refer to the figure on the next page.**

The servo control functions of the **DC2** have been designed to use biphase (quadrature) incremental encoders as position feedback. The circuits on the **DC2** can accept either differential or single ended signals for the two encoder phases and the optional index signal.

To use differential encoder phase signals on axis 1, leave the pins of jumpers JP15 and JP16 open. Connect encoder phase A signals to pins 2 and 5 of connector J1. Connect encoder phase B signals to pins 10 and 13 of connector J1.

To use single ended encoder phase signals on axis 1, connect pins 1 and 2 of jumper JP15, and pins 1 and 2 of jumper JP16. Connect encoder phase A signal to pin 2 of J1. Connect encoder phase B signal to pin 10 of J1.

To use differential index signal on axis 1, leave the pins of jumper JP17 open. Connect the index signals to pins 6 and 14 of J1. The index signal on pin 6 should go high, and the signal on pin 14 low, when the index mark is hit.

To use an active high single ended index signal on axis 1, connect pins 1 and 2 of jumper JP17. Connect the signal to pin 6 of J1. To use an active low single ended index signal on axis 1, connect pins 2 and 3 of jumper JP17. Connect the signal to pin 14 of J1.

To use differential encoder phase signals on axis 2, leave the pins of jumpers JP19 and JP20 open. Connect encoder phase A signals to pins 2 and 5 of connector J2. Connect encoder phase B signals to pins 10 and 13 of connector J2.

To use single ended encoder phase signals on axis 2, connect pins 1 and 2 of jumper JP19, and pins 1 and 2 of jumper JP20. Connect encoder phase A signal to pin 2 of J2. Connect encoder phase B signal to pin 10 of J2.

To use differential index signal on axis 2, leave the pins of jumper JP18 open. Connect the index signals to pins 6 and 14 of J2. The index signal on pin 6 should go high, and the signal on pin 14 low, when the index mark is hit.

To use an active high single ended index signal on axis 2, connect pins 2 and 3 of jumper JP18. Connect the signal to pin 6 of J2. To use an active low single ended index signal on axis 2, connect pins 1 and 2 of jumper JP18. Connect the signal to pin 14 of J2.

As a final step in connecting the servo encoders, 4 jumpers are provided to select the encoder phasing of the two axes. In phasing each encoder there are two objectives. First, to get the **DC2** controller to drive the motor in order to hold it at a desired position. Second, to cause the
controller to count encoder pulses in the right direction (eg. clockwise rotation results in more positive position count).

The jumpers that select the encoder's phasing will allow at least one of the objectives to be met. In order to accomplish both, it may be necessary to switch the jumpers, reverse the leads on the motor, and/or issue the **DC2** phasing command. At this point, it is only necessary to install the jumpers in one of two positions in order for the controller to read encoder positions.

For standard encoder phasing on axis 1, (default setting) connect pins 1 and 2 of jumper

JP33 and pins 1 and 2 of jumper JP34. For reverse phasing, connect pin 1 of JP33 to pin 1 of JP34 and pin 2 of JP33 to pin 2 of JP34.

For standard encoder phasing on axis 2, (default setting) connect pins 1 and 2 of jumper JP35 and pins 1 and 2 of jumper JP36. For reverse phasing, connect pin 1 of JP35 to pin 1 of JP36 and pin 2 of JP35 to pin 2 of JP36.

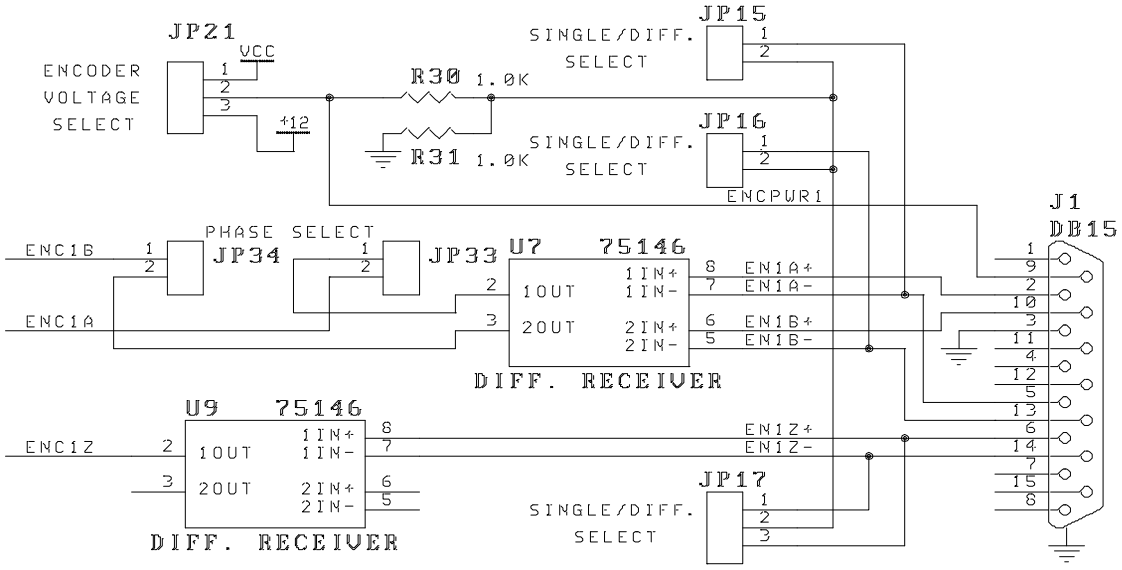


Figure 5: DC2 encoder circuit (typical)

## 2.3 COARSE HOME AND LIMIT SWITCH CONNECTIONS

The **DC2** includes a coarse home input and two limit switch inputs for each servo axis. The coarse home input is used to qualify the encoder index pulse to when initializing (homing) an axis. The limit switch inputs are used to signal the controller when an axis has exceeded acceptable travel limits. These three inputs for each axis are optional and may not be required in some applications. In these cases, leave the respective inputs unconnected.

These inputs to the **DC2** are jumper selectable to be received by either 74LS14 TTL input buffers or 4N29 optical isolators. When using the TTL input buffers, the inputs are low active (less than .7 vdc). When using the optical isolators, the signal should be between 12 and 24 vdc when active. The ground for the optical isolator power supply should be connected to J1 pin 8. The signal must be capable of sourcing at least 5 ma. Note that other voltages can be used, please contact the factory for details. To change the active level of an input using the Limit Mode command refer to section 10.1 (Parameter Setup Commands). The drawing below illustrates the two options for interfacing external switches/sensors to a optically isolated DC2 input.

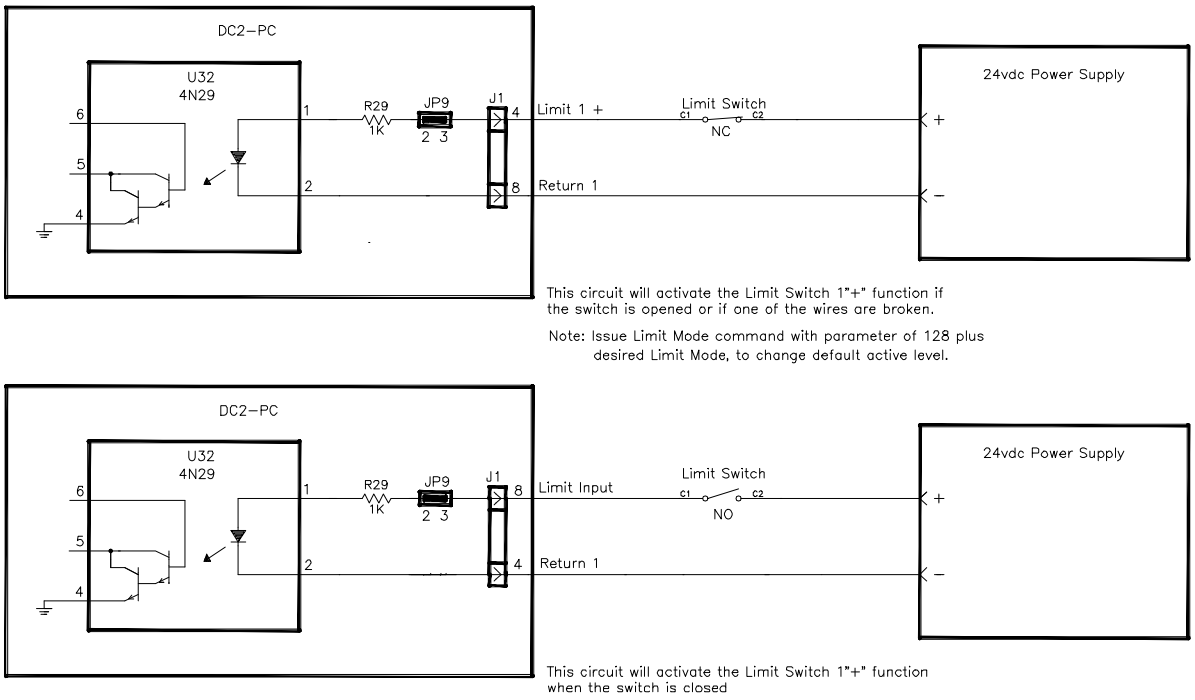


Figure 6: Optically isolating external switches/sensors

To assist with switch/sensor connection please refer to the schematic on the next page.

To use a TTL level buffer on the coarse home input of axis 1, connect pins 1 and 2 of jumper JP8. Connect the input signal to pin 7 of J1.

To use an optical isolator on the coarse home input of axis 1, connect pins 2 and 3 of jumper JP8. Connect the input signal to pin 7 of J1 and pin 8 of J1 to the signal return (ie. signal supply ground).

To use TTL level buffers on the limit switch inputs of axis 1, connect pins 1 and 2 of jumper JP9 for limit+, and pins 1 and 2 of jumper JP10 for limit-. Connect the limit+ signal to pin 4 of J1 and the limit- signal to pin 12 of J1.

To use optical isolators on the limit switch inputs of axis 1, connect pins 2 and 3 of jumper JP9 for limit+, and pins 2 and 3 of jumper JP10 for limit-. Connect the limit+ signal to pin 4 of J1 and the limit- signal to pin 12 of J1. Connect pin 8 of J1 to the signal return.

To use a TTL level buffer on the coarse home input of axis 2, connect pins 1 and 2 of jumper JP11. Connect the input signal to pin 7 of J2.

To use an optical isolator on the coarse home input of axis 2, connect pins 2 and 3 of jumper JP11. Connect the input signal to pin 7 of J2 and pin 8 of J2 to the signal return.

To use TTL level buffers on the limit switch inputs of axis 2, connect pins 1 and 2 of jumper JP12 for limit+, and pins 1 and 2 of jumper JP13 for limit-. Connect the limit+ signal to pin 4 of J2 and the limit- signal to pin 12 of J2.

To use optical isolators on the limit switch inputs of axis 2, connect pins 2 and 3 of jumper JP12 for limit+, and pins 2 and 3 of jumper JP13 for limit-. Connect the limit+ signal to pin 4 of J2 and the limit- signal to pin 12 of J2. Connect pin 8 of J2 to the signal return.

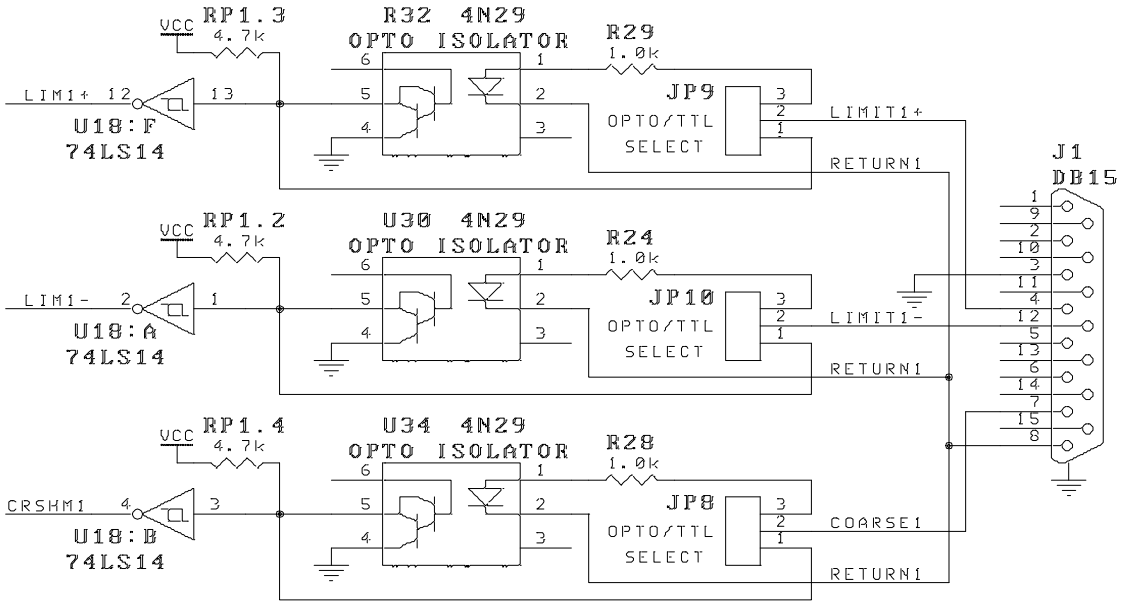


Figure 7: DC2 switch circuit (typical)

## 2.4 DIGITAL I/O CONNECTIONS

The **DC2** provides an interface for as many as sixteen general purpose digital I/O (TTL level) channels. Each channel is individually programmable as input or output using high or low true logic. These signals are available on connector J3 (refer to appendix B for pinout). The maximum current sink/source capability of each channel is 1 ma. Each channel has a 10K ohm pull-up resistor to 5 vdc.

The digital I/O interface utilizes a 26 pin (.025" square posts, .100" inch centers) shrouded header with a center polarizing tab. The typical cabling solution is to use a 26 pin Female IDC (ribbon cable) mating connector. One approved source for this connector is available from Circuit Assembly (P/N CA-26IDS2-F-SPT). For applications that require point to point cabling (20 - 24 awg wire), female crimp pins and center polarizing pin housing are available from AMP (crimp pin P/N 87523-6, pin housing P/N 102387-6, hand tool P/N90202-2)

**Optical Isolation and non TTL level digital I/O interfacing**

**Please refer to the figures on the next two pages.**

Some applications may requiring the **DC2** Digital I/O to interface to other than low current TTL level external devices. The BFO22 relay rack interface board (available from PMC) provides a convienent means of connecting the **DC2**'s I/O channels to a 16PBH relay rack available from Opto22. The 16PBH accepts up to 16 optically isolated input or output modules for interfacing with external electrical systems. Using the 16PBH and a BFO22, each installed Opto 22 module will be connected to one of the **DC2**'s digital I/O channels.

For installation, the BFO22 plugs directly into the 16PBH and then connects to the **DC2** via a 26 conductor ribbon cable with IDC connectors. Note that the relay rack modules are numbered 0 through 15, while the **DC2** I/O channels are numbered 1 through 16.

Although the relay rack has screw terminals for connecting a logic supply, it is not necessary to make this connection. By installing a shorting block on jumper JP17 of the BFO22, the logic supply of the **DC2** will be supplied to the 16PBH.

For each I/O module position on the 16PBH relay rack, there is a corresponding jumper on the BFO22. By placing a shorting block in one of two positions on each of these jumpers, the BFO22 circuitry is configured for connection to either input or output modules. The BFO22 jumper JP1 is used to configure the circuitry for module position 0, jumper JP2 is for module position 1, and so on. If a module position has an output relay, a shorting block should be placed on pins 1 and 2 of the corresponding jumper. Conversely, if a module position has an input signal conditioner, the shorting block should be placed between pins 2 and 3. Note that pin 1 of each jumper is the one closest to the jumper label.

**Optical Isolation and non TTL level digital I/O interfacing (cont.)**

Before setting the individual I/O channels as inputs or outputs, a global **C**hannel **L**ow command should be issued to the **DC2**. This will cause "normally open" relays to turn on when the **C**hannel o**N** command is issued, and off when the **C**hannel o**F**f command is issued. For input modules, the **T**ell **C**hannel command will return a "1" when the signal is present, and a "0" when it is not.

By default, all I/O channels on the **DC2** are changed to inputs during power-up or reset. For

input modules installed in the relay rack, no further configuration is necessary. In order to configure any channels for output modules, individual **C**hannel ou**T** command should be issued to the **DC2**.

Example: Channels 1 and 2 are inputs, channels 3 and 4 are outputs, use the following commands to configure them:

**CL**,**CT**3,**CT**4

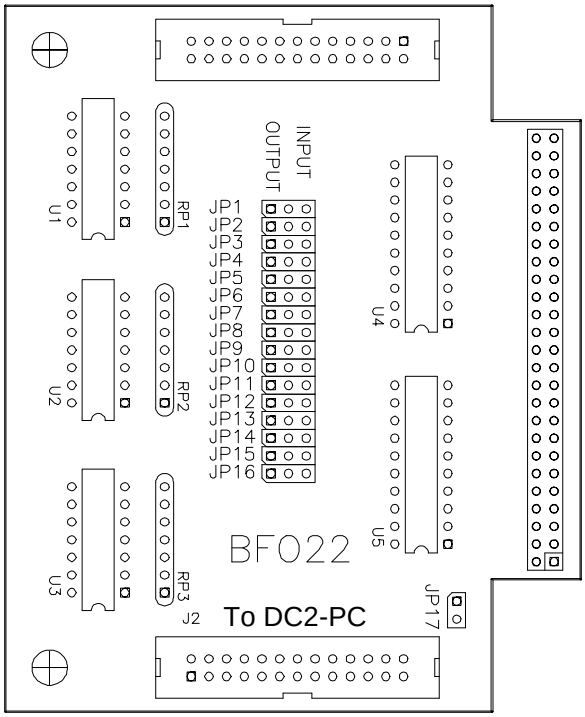


Figure 8: BF022 Relay Rack interface card
for OPTO 22/GRAYHILL

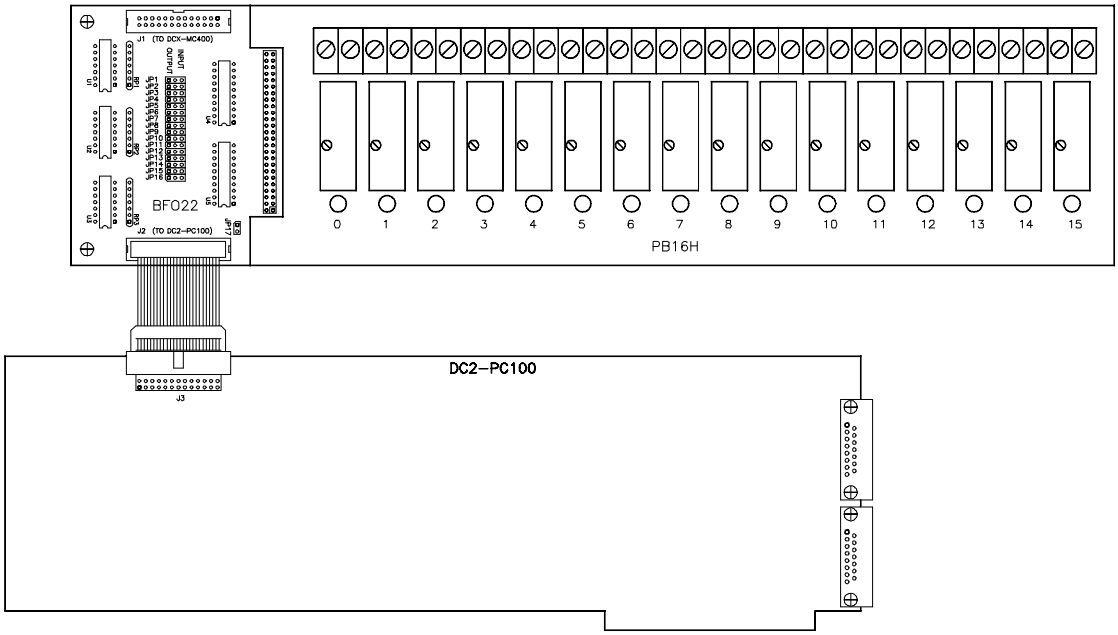


Figure 9: DC2 Digital I/O to Relay Rack Interface

## 2.5 A-D CHANNEL CONNECTIONS

The **DC2** provides an interface for as many as four A-D input channels (8 bit) channels. These signals are available on connector J3 (refer to appendix B for pinout).

The A-D interface utilizes a 26 pin (.025" square posts, .100" inch centers) shrouded header with a center polarizing slot. The typical cabling solution is to use a 26 pin Female IDC (ribbon cable) mating connector. One approved source for this connector is available from Circuit Assembly (P/N CA-26IDS2-F-SPT). For applications that require point to point cabling (20 - 24 awg wire), female crimp pins and center polarizing pin housing are available from AMP (crimp pin P/N 87523-6, pin housing P/N 102387-6, hand tool P/N90202-2)

The maximum voltage range for the A-D channels is 0 to +5 vdc. With jumper JP14 set to pins 2 and 3, an on board precision +5vdc regulator is used as a reference for the A-D conversion. The eight bit resolution of the A-D channels will result in a conversion of approximately .019 volts per DAC unit. The **T**ell **A**nalog (**TA*x***) command, where x = the required channel, is used to report the value of the analog channel. Setting jumper JP14 to pins 1 and 2 (default configuration) configures the **DC2** for scaling of the A-D inputs. The **DC2** will scale the A-D inputs to whatever voltage level is supplied on connector J3 pin 23 (Analog Reference). The drawing below is an example of the setup required to scale an A-D channel for 8 bit resolution over a 2 volt range.

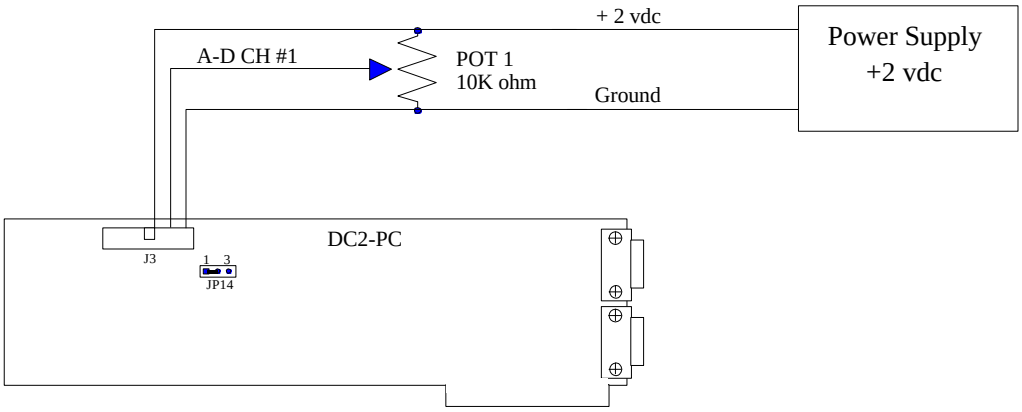


Figure 10: A-D interface interconnect example

External Analog reference = 2.0 vdc

Analog to digital conversion = Analog reference / Resolution = ____ volts per DAC unit
2.0 vdc / 256 = .0078 volts per DAC unit

## 2.6 RS-232 INTERFACE CONNECTIONS

The DC2 RS-232 interface is designed to connect directly to a nine pin com port on a IBM-PC,XT,AT or compatible computer. The RS-232 interface connector J4 is a 10 pin (.025" square posts, .100" inch centers) dual row shrouded header with a center polarizing slot. The recommended cabling solution is to use a 10 conductor ribbon cable with:

| Cable end to: | Connector type |
|---|---|
| DC2 end of the cable | 10 pin, dual row, female IDC connector with center polarizing tab |
| PC computer end of the cable | 9 pin, female, DB9 IDC connector |

Note: For applications where a nine pin com port is not available it is recommended that a DB25 to DB9 conversion cable **not be used**. Please refer to the following special cabling descriptions.

**9 pin PC com port three wire interface**

The **DC2** can be interfaced to a 9 pin PC com port using a three wire interface but hardware handshaking will not be supported. This configuration **also** requires that a wire be added to the **DC2** board from **J4 pin 7 to J4 pin 8**. Make a cable using: discrete wires, a male DB9 connector, and a pin housing that will mate to a 10 pin dual row shrouded header. Make the connections as shown below:

| Communication Device DB9 | DC2-PC J4 conn. 10 pin connector |
|---|---|
| 2 | 3 |
| 3 | 5 |
| 5 | 9 |

**25 pin PC com port interface**

The **DC2** can be interfaced to a 25 pin PC com port two different ways. Please note that only the first cable configuration will support Hardware Handshaking.

Make a cable using: discrete wires, a female DB25 connector, and a pin housing that will mate to a 10 pin dual row shrouded header. Make the connections as shown below:

| Communication Device DB25 | DC2-PC J4 conn. 10 pin connector |
|---|---|
| 2 | 5 |
| 3 | 3 |
| 4 | 4 |
| 5 | 6 |
| 6 | 2 |
| 7 | 9 |
| 8 | 10 |

20 ..... 7
22 ..... 8

This cable connections described below will not support hardware handshaking. This configuration **also** requires that a wire be added to the **DC2** board from **J4 pin 7 to J4 pin 8**. Make a cable using: discrete wires, a male DB25 connector, and a pin housing that will mate to a 10 pin dual row shrouded header. Make the connections as shown below:

| Communication Device DB25 | DC2-PC J4 conn. 10 pin connector |
|---|---|
| 2 | 3 |
| 3 | 5 |
| 5 | 9 |

On the DC2-PC J4 connector jumper J4-7 to J4-8.

# 3.0 COMMUNICATIONS INTERFACES

In order to send commands to the **DC2** board, either of two communication interfaces are available. Both ASCII and Binary interfaces are supported by the DC2 for communication with IBM-PC/XT/AT or compatible computers.The ASCII interface, when used with the DC2 utilities, is powerful yet easy to use. The Binary interface (see appendix C) is designed for sophisticated applications programmed in high level languages. The Binary interface allows the PC host to write binary formatted commands directly into the **DC2**'s command buffer. This provides the fastest PC to **DC2** command throughput. See appendix C for a full description of the Binary Mode command interface. The RS-232 serial interface supports communicating to other types of computers or display terminals.

Commands sent to the **DC2** through either of the ASCII communication interfaces must be followed by a carriage return (ASCII 13). There should be no line feeds (ASCII 10) included at the end of command lines. When multiple commands are grouped on a single line they should be separated by commas ",".

## 3.1 PC INTERFACE

If the **DC2** board is installed in an PC/XT/AT compatible host computer, communications utilities are provided for both Windows and DOS based installations.

### Windows Terminal Emulator

For systems using Microsoft Windows, after installing the **PMC WINDOWS UTILITIES**, the user can select **PMC WinControl**. This Windows based Terminal Emulator implements communications with the **DC2** through the host interface. In addition to sending command lines typed at the keyboard to the **DC2** and displaying responses on the host's monitor, this program can also be used to send a text file containing **DC2** commands to the controller. This file download feature supports a range of capabilities from initializing motor parameters each time the **DC2** is powered up, to sending a complete ASCII format motion control program. Simply store the command lines in a file using a text editor, then select **FILE** and **OPEN** from the Windows menu. Enter the name of the file to be sent to the **DC2** and select **OK** from the Windows dialog box.

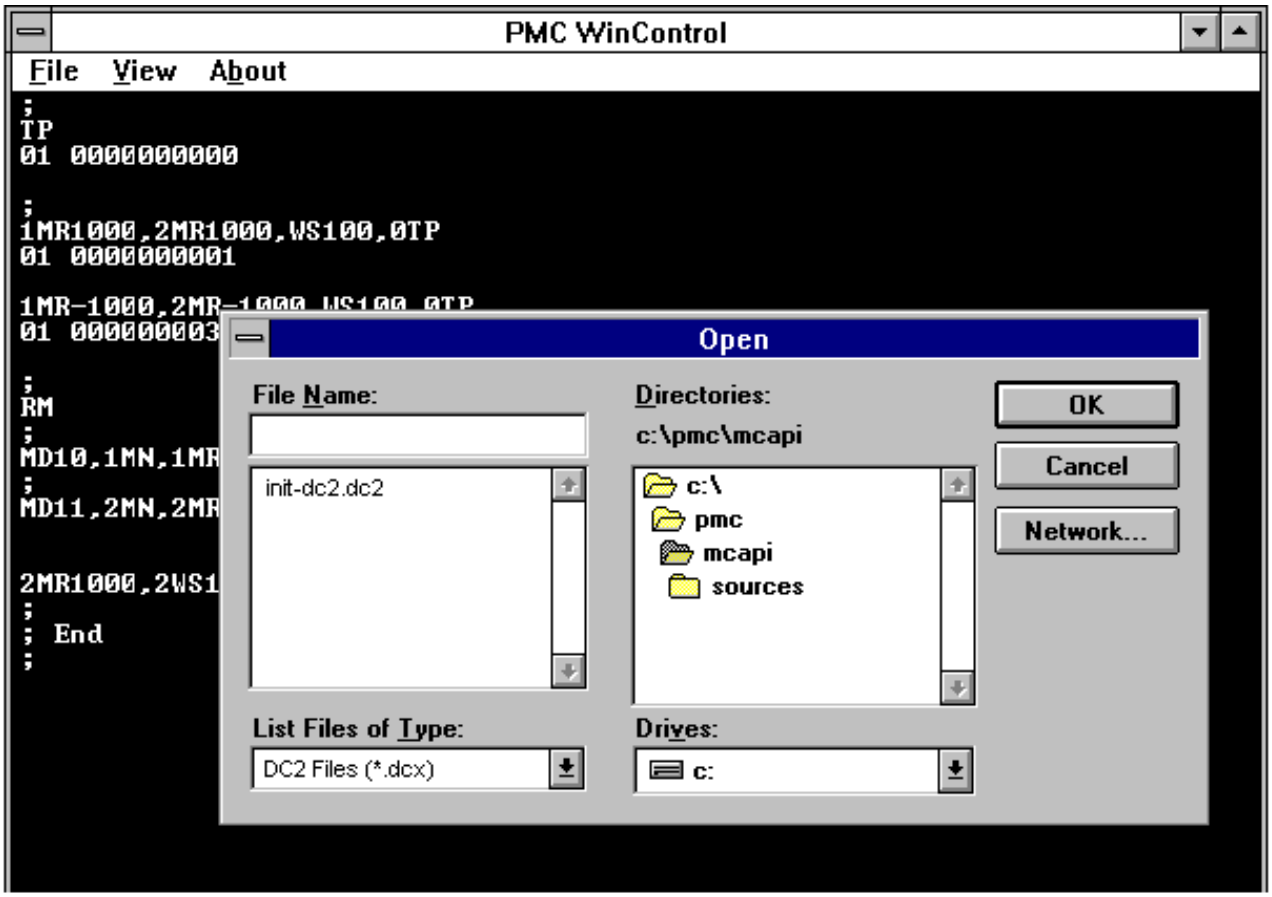


Figure 11: PMC WinControl; a Windows based motion control command interface

**DOS Terminal Emulator**

For DOS based systems, the Terminal Emulator found on the "**DC2** DOS Utility Disk" implements communications with the **DC2** through the host interface. This program can be run by inserting the "DC2 DOS Utility Disk" in a disk drive and typing DC2CNTRL. In addition to sending command lines typed at the keyboard to the **DC2** and displaying responses on the host's monitor, this program can also be used to send a text file containing **DC2** commands to the controller. This file download feature supports a range of capabilities from initializing motor parameters each time the **DC2** is powered up, to sending a complete ASCII format motion control program. Simply store the command lines in a file using a text editor and then include the name of the file when invoking the DCXCNTRL program. For example, if the **DC2** commands were stored in file called INIT.DC2, type:

DC2CNTRL INIT.DC2.

If the application requires a program running on the host to issue commands to the **DC2**, the programmer should examine the Windows utilities and DOS interface libraries that are supplied on the utility tools disks. Device drivers and Interface libraries for popular programming languages (Visual C, Visual BASIC, 'C', Pascal and BASIC) can be found on the disks. The 'READ.ME' files on the disks list what files are associated with each driver or interface library.

The **DC2** utilities and interface libraries are implemented using the '**DC2** Binary Command Interface'. This interface uses binary formatted commands and replies and provides the most efficient means for the host to communicate with the **DC2**. It is described in full detail in appendix 'C' of this manual. High level language ('C' and Basic) program examples utilizing the binary interface can be found in appendix D and E. In some situations, it is preferable for the host to send commands to the **DC2** in an ASCII character format, and for the board to send replies as ASCII characters. This can be done through the 'ASCII Command Interface' of the **DC2**. This interface is described below.

**ASCII interface**

If the terminal emulator utilities described above satisfies the users requirements, it is not necessary to read or understand the remainder of this section. What follows is a description of how the PC communications interface of the **DC2** is implemented. It is provided for those users wanting to write their own PC to DC2 interface programs.

When a **DC2** board is plugged into an IBM-AT or compatible, four 8 bit registers located on the **DC2** appear in the PC's address space. It is through these registers that the PC is able to monitor and communicate with the **DC2.** It is possible to cause these registers to appear in either the I/O (default configuration) or memory space of the PC, depending on the placement of the 74LS688 IC in socket U14. There are four rows of pin sockets at this location. The IC can be installed in one of two positions labeled U14A and U14B. For I/O space addressing place the IC in U14B; for memory space addressing place the IC in U14A. Note that jumpers JP31 and JP32 require jumper blocks depending on the placement of the IC also. See the appendix covering jumpers for full details.

The base address where the registers are accessed is determined by the setting of 8 jumpers, JP23 through JP30 on the **DC2**. When the **DC2** is shipped from the factory in the standard configuration, the base location of these registers will be at 300 hex in the I/O address space. This is accomplished by connecting pins 1 and 2 of jumpers JP23 and JP24, connecting pins 2 and 3 of jumpers JP25 - JP31 and 32, and by installing the 74LS688 in the U14B position. Please refer to the following table for **DC2** board addressing:

| Board# | JP23 | JP24 | JP25 | JP26 | JP27 | JP28 | JP29 | JP30 | Base Add. |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1-2 | 1-2 | 2-3 | 2-3 | 2-3 | 2-3 | 2-3 | 2-3 | 300h |
| 1 | 1-2 | 1-2 | 2-3 | 2-3 | 2-3 | 2-3 | 2-3 | 1-2 | 304h |
| 2 | 1-2 | 1-2 | 2-3 | 2-3 | 2-3 | 2-3 | 1-2 | 2-3 | 308h |
| 3 | 1-2 | 1-2 | 2-3 | 2-3 | 2-3 | 2-3 | 1-2 | 1-2 | 30Ch |
| 4 | 1-2 | 1-2 | 2-3 | 2-3 | 2-3 | 1-2 | 2-3 | 2-3 | 310h |
| 5 | 1-2 | 1-2 | 2-3 | 2-3 | 2-3 | 1-2 | 2-3 | 1-2 | 314h |
| 6 | 1-2 | 1-2 | 2-3 | 2-3 | 2-3 | 1-2 | 1-2 | 2-3 | 318h |
| 7 | 1-2 | 1-2 | 2-3 | 2-3 | 2-3 | 1-2 | 1-2 | 1-2 | 31Ch |
| 15 | 1-2 | 1-2 | 2-3 | 2-3 | 1-2 | 1-2 | 1-2 | 1-2 | 33Ch |

The offset of the registers from the base address are as follows:

| OFFSET | READ | WRITE | DESCRIPTION |
|---|---|---|---|
| 0 | X | | DATA IN (into **DC2**) |
| 0 | | X | DATA OUT (out of **DC2**) |
| 1 | X | | STATUS |
| 1 | | X | ATTENTION |

The bit assignments of the PC bus communication Status register are as follows:

| BIT | DESCRIPTION |
|---|---|
| 0 | BREAK POINT AXIS 1 |
| 1 | BREAK POINT AXIS 2 |
| 2 | TRAJECTORY COMPLETE AXIS 1 |
| 3 | TRAJECTORY COMPLETE AXIS 2 |
| 4 | SERVO OR COMMAND ERROR |
| 5 | BUSY EXECUTING COMMAND |
| 6 | READY WITH OUTPUT DATA |
| 7 | READY FOR INPUT DATA |

Note that the Status register can be read by the PC at any time without having any effect on the **DC2**. However the Data Out and Data In registers are wired to clear bits 6 and 7 of the Status register respectively. For this reason, the Data registers should only be read or written to once per data byte transferred.

The ASCII protocol is used when the **DC2** is in "Decimal" or "Hexadecimal" mode. This protocol uses ASCII coded characters to issue commands to the **DC2** board. Any replies

from the **DC2** will also be ASCII coded characters. In order to send ASCII characters to the board, first read the Status register, if the Ready For Input flag is set, write the first character to the Data In register. If the flag is not set, the **DC2** has not read data previously written to the register. The PC should continue checking the status until the flag is cleared, and then write the character. This procedure should be repeated for each charater to be sent to the DCX.

In order to receive ASCII characters from the board, read the Status register. If the Ready With Output flag is set, a valid character can be read from the Data Output register. If the flag is not set, the **DC2** has not placed new data in the register. In this case the PC can continue checking the status.

## 3.2 RS-232 SERIAL COMMUNICATIONS INTERFACE

The serial communications interface has been designed to use either hardware handshaking or XON-XOFF protocol for its operation. Hardware handshaking can be enabled or disabled by use of the **HN** and **HF** commands. XON-XOFF protocol can be enabled or disabled by use of the **XN** and **XF** commands.

The configuration of the communicating device should be set to 9600 baud, 8 data bits, one stop bit and no parity. The **DC2** defaults to this configuration after reset. The communication software should also be set for carriage return only at the end of each command string.

The **DC2** controller echos a line feed character after receiving a command when using the RS-232 interface, this is done to properly format the display.

Note: Input characters are only echoed through the serial interface when enabled by the **E**cho o**N** (**EN**) command.

# 4.0 DC2 OPERATION

In order for the **DC2** to perform any operation, it must first receive a command. This command can be any one of the predefined commands described in section 9 later in this manual. If the command is sent to the **DC2** through one of the available communication interfaces, it will be executed as soon as it is received. Alternatively, the command can be stored in the **DC2**'s own memory and caused to be executed by external signals. But no matter where the command originates from, it will be executed by the **DC2** in the same way. Because the commands consist of standard alphanumeric characters and are built with two key characters from the description of the operation (eg. "**MR**" for **M**ove **R**elative), they are easy to use and remember.

In order to become familiar with the operation of the **DC2**, it is suggested that the user have a means of sending commands to the board by typing at a keyboard. For PC based systems two interface utilities are provided; **DC2CNTRL** for DOS systems or **PMC WIN CONTROL** for Windows systems. Both utilities allow commands to be entered at the keyboard and responses will be displayed on the screen.

For applications utilizing the RS-232 serial communications interface, PC based programs (like PROCOMM, HyperTerminal, or Windows Terminal) allow the computer to emulate a terminal, providing a convenient means of sending commands to the **DC2**.

Once power has been applied to the **DC2**, you can begin issuing commands to it. A good command for verifying correct operation of the communications link is the **VE**rsion command. Type '**VE**' followed by the return key. In response, the controller should display the following:

> DC2-PC140 Motion Controller Ver. ___ Rev ____
> COPTWRIGHT © PRECISION MICRO CONTROL CORPORATION 1990-1992
> ALL RIGHTS RESERVED

Note that each character typed at the keyboard should be echoed to your display (this requires Echo oN which is the default setting). If you enter an illegal command, the **DC2** will respond with a question mark, followed by an error code. For interpretation of the error code, see appendix F of this manual. On receiving this response, you should re-enter the the entire command string. If you make a mistake in typing before the return key is pressed, the backspace can be used to correct it.

## 4.1 MACRO COMMANDS

One of the most powerful tools available to the **DC2** user is the use of macro commands. This simply means assigning one (macro) command to represent a whole string of commands.

For example, "1**MR**1000,**WS**250,1**MR**-1000,**WS**250" may be entered as a command string. If this sequence were to represent a frequently desired motion for the system, it could be defined as a macro command. This is done by inserting a **M**acro **D**efinition (**MD**) command as the first command as shown below:

**MD**3,1**MR**1000,**WS**250,**MR**-1000,**WS**250

This will define the command string as macro number 3. Whenever it is desired to perform this motion sequence, just issue the **M**acro **C**all command "**MC**3".

There are 8192 bytes of the **DC2**'s memory allocated to macro storage. Each command stored in a macro requires 6 bytes of storage, plus an overhead of 1 byte per macro. The example given for macro number 3 would use 4 x 6 + 1, for a total of 25 bytes. Note that the **MD** command itself is not stored with the macro.

A built in function of the **DC2** is that it will execute macro 0 on power-up or reset. After the **R**eset **M**acros command is issued, all macros are undefined. The **DC2** recognizes this and skips the execution of macro 0. But once macro 0 has been defined by the user with the **M**acro **D**efinition command, it will be executed after any reset. This is useful for automatically configuring the **DC2** each time it is reset. It also allows the **DC2** to be installed without a host computer, operating totally on it's own. The complexity of what macro 0 can accomplish is limited only by the user's ability to write macros.

To verift that a macro was stored correctly isssue the Tell Macro (TMn) command where n equals macro number. To report all macro's issue the Tell Macro command with a parameter of '-1'.

To clear all macros from memory issue Reset Macro (RM) command.

Execution of a macro can be halted by entering the 'escape' key or issuing the ASCII 'escape' code.

## 4.2 MOVING MOTORS

Once you are satisfied that the communication interface is correctly conveying your commands and responses, you are ready to check the servo interface. When the **DC2** is powered up or reset, all servos will be in the "motor off" state. In this state, there should be no drive current to the motors. When using external amplifiers to drive servos, it is possible for a small offset voltage to be present. This is usually too small to cause any motion, but some systems have so little friction that a few millivolts can cause them to drift objectionably. If this is the case, the "null" voltage can be minimized by adjusting the offset adjustment potentiometers (POT1 & POT2) located on the board.

Before a servo can be controlled by the **DC2**, certain internal parameters must be set. These include filter gains, acceleration, deceleration and maximum velocity. The **DC2** automatically sets the parameters to default values on power-up or reset. Appendix F contains the default settings of these parameters.

Assuming that the default servo parameters are acceptable, the servos can be changed to the "on" state. To enable all axes, issue the **M**otor o**N** command by entering **MN** followed by (return). To enable only 1 axis, specify which one before the command mnumonic. For example, 1**MN** will only turn axis 1 "on".

After turning a particular axis "on", it should hold steady at one position without moving, and the **T**ell **T**arget (**TT**) and **T**ell **P**osition (**TP**) commands should report the same number. There are several commands which are used to cause motion, including the **M**ove **A**bsolute(**MA**), **M**ove **R**elative(**MR**), **M**ove to **P**oint(**MP**), and **G**o **H**ome (**GH**) commands. To move axis 2 by 1000 encoder counts, enter 2MR1000 and a carriage return. If the axis is in the "**M**otor o**N**" state, it should move in the direction defined as positive for that axis. To move back, enter 2**MR**-1000 and a carriage return.

With the **DC2** controller, it is possible to group together several commands. This is not only useful for defining a complex motion which can be repeated by a single keystroke, but is also useful for synchronizing multiple motions. To group commands together simply place a comma between each command, pressing the return key only after the last command.

A repeat cycle can be set up with the following command string:

1**MR**1000,**WS**32,**MR**-1000,**WS**32,**RP**6(return)

This command string will cause axis 1 to cycle 7 times. (One time by command, repeated 6 more times.) The **W**ait for **S**top (**WS**) commands are required so that the motion will be completed before the return motion is started. The number 32 following the **WS** command specifies the number of milliseconds to wait after the trajectory complete flag in the servo status has been set. This allows some time for the mechanical components to come to rest, and to reduce the stresses that could occur on them if the motion were reversed instantaneously. Notice that the "axis number" need be specified only once with the first servo command in the string.

A more complex cycle could be set up involving multiple axes. In this case, the axis to be affected by the command is assumed to be the last one specified in that command string. Whenever a new command string is entered, the axis is assumed to be 0 (all) until one is specified.

Entering the following command:

1**MR**1000,2**MR**-500,0**WS**300,1**MR**1000,2**MR**500,0**WS**300,**RP**4 <return>

followed by return will cause axis 1 to move in the positive direction and axis 2 to move in the negative direction. When both axes have stopped moving, the **WS** command will cause a 300 millisecond delay after which the remainder of the command line will be executed. The **RP4** command will cause the entire motion sequence to be repeated 4 more times.

After completing this complex motion 5 times, it can be repeated another 5 times by simply entering a return character. Command strings will be retained by the controller until another command string is entered. This comes in very handy for observing the position display during a move. If you enter:

1**MR**1000 <return>
1**TP** **<**return>
<return>
<return>

the **DC2** will respond with a succession of numbers indicating the position of the axis at that time.

Another way to monitor the progress of a movement is to use the RP command without a value. This will repeat until stopped by the operator. It may be stopped by pressing the "escape" key on the keyboard or sending an ASCII escape code.

When the following is entered:

1**MR**10000 <return>
1**TP**,**RP** <return>

the position will be displayed continuously until it is stopped. The user may also specify the number of times the sequence is to be repeated, or it may be repeated indefinitely. Pauses lasting from 1 millisecond to over 65 seconds may be inserted between elements of the sequence at the user's option. The sequence may be repeated using the repeat (**RP**n) command, or it can be repeated any number of times by simply entering the carriage return when desired.

While the **DC2** is executing commands, the operation can be halted by pressing the escape key. Commanded motion will continue, but no additional commands will be executed. The execution of commands can be "paused" by pressing the space bar. To resume command execution the space bar is pressed again. While a command is paused, it can be aborted by pressing the escape key.

# 5.0 SERVO OPERATION

**Encoder checkout**

After connecting the servo's encoder, the **DC2** should report its position each time the **T**ell **P**osition (***a*TP**) command is issued.

Example:

**TP**
01 0

If the servo is manually turned in either direction, the position reported should increment or decrement accordingly.

Example:

**TP**
01 250

Next the output signal from the module can be connected to the external amplifier. Use the **S**top on **E**rror (**SE**) command to set the maximum following error of the servo. This will stop the servo and set the motor error bit in the servo status word in the case that it is 'reverse phased'.

Example:

1**SE**1000

Next set the proportional gain of the servo loop to a very low value.

Example:

1**SG**50

Now turn the specific axis on using the **M**otor o**N** (**MN**) command.

Example:

1**MN**

If the Servo motor begins to oscillate reduce the SG value by 50%.

Manually move the servo from its present position. If the motor is properly phased, it should resist movement away from its current position. Be aware that if the servo is reverse phased, it may jump to full torque or speed. In this case, the **S**top on **E**rror (1SE1000) command should stop the servo (and set the Motor Error bit of the status word) once the difference between the current position and the desired position is greater than 1000 encoder counts. If this happens, the servo phasing is changed by either:

1) Issuing the **PH**ase command to the axis with a parameter of 1
2) Reversing the encoder inputs
3) Reversing the motor leads

**Tuning the servo**

The following instructions describe the command line method of servo tuning. If a PC/AT

computer with Windows is available, the DC2 Windows Tunung Utility will greatly simplify the tuning process.

Before tuning a servo the torque limit (***a*SQ*n***) must be set to match the **DC2** output configuration. The default value (*n* = 127) is set for a DC2-PC140 (using the on board PWM driver). If the output configuration of the axis being tuned is a -/+ 10 analog control signal, the torque limit should be set to *n* = 2047.

1SQ2047

The proportional gain controls the 'stiffness' or restoring force of a servo. Repeatedly issue Set proportional Gain (***a*SG*n***) commands, increasing the value *n* by 50% until the the motor begins to oscillate.

1SG75
1SG112
1SG168

The oscillation can be dampened by adding derivative gain to the servo loop. This is done with the **S**et **D**erivative command. Repeatedly issue Set Derivative (***a*SD*n***) commands, increasing the value *n* by 50% until the the motor stops oscillating.

1**SD**100
1SD125
1SD187

For applications with high inertia load, increasing the derivative sampling period (aFRn) can eliminate oscillation without overdampening the responsivness of the servo (see the set derivative sampling period (***a*FR*n***) command in section 9.1)

In order to decrease static error (after a move is completed), integral gain can also be added to the servo loop using the **S**et **I**ntegral gain (**SI**) and set Integration Limit (IL) commands. The set Integration **L**imit (**IL**) command limits the level of power that the integral gain (SI) can use to reduce position error. Issuing the set Integration Limt (IL) command with a non zero parameter is required for the Set Integral gain (SI) to have any effect.

Example:
1**IL**20
1**SI**5

If the servo is being driven by a velocity mode amplifier, the velocity feed-forward term should be set with the **V**elocity **G**ain (**VG**) command. Consult the factory for assistance.

**Moving Motors**

Once the servo is properly "tuned", the operating mode is selected. Use the **P**osition **M**ode (**PM**) command for point to point motion.

Example:
1**PM**

The maximum velocity and acceleration can now be set using the **S**et **V**elocity (**SV**), **S**et **A**cceleration (**SA**) and **D**eceleration **S**et (DS). These commands should always be used before issuing any motion commands to the axis. If this is not done, the motor will not move

as expected. The parameters for each of these commands are in units of encoder counts per 1 msec. (velocity command), or encoder counts per 1 msec. per 1 msec. (acceleration and deceleration commands).

Example:

1**SV**100000
1**SA**100000
1**DS**100000

The trajectory parameters for a desired move are set as follows. If, for example, connected to axis one is a 500-line shaft encoder, and the motor must accelerate at one revolution per second per second until it is moving at 600 rpm, and then decelerate to a stop at a position exactly 100 revolutions from the start, one would calculate the trajectory parameters as follows:

Setting axis #1 velocity (1SV*n*) parameter
let P = target position (units = encoder counts)
let R = encoder lines * 4 (system resolution)
then R = 500 * 4 = 2000

and P = 2000 * desired number of revolutions
P = 2000 * 100 revs = 200,000 counts (value to load)

let V = velocity (units = counts/sample)
let T = sample time (seconds) = 1 ms
let C = conversion factor = 1 minute/60 seconds
then V = R * T * C * desired rpm
V = 2000 * 1E-3 * 1/60 * 600 rpm
V = 19.99 counts/sample
V (scaled) = 19.99 * 65,536 = 1310064.64
V (rounded) = 1310065 (value to load)

**1SV1310065** ;DC2 set velocity command

**Setting axis #1 acceleration (1SA*n*) and deceleration (1DS*n*) parameters**
let A = acceleration and deceleration
A = R * T * T * desired acceleration
then A = 2000 * 1E-3 * 1E-3 * 1 rev/sec/sec
A = 2E-3 counts/sample/sample
A = (scaled) = 2E-3 * 65,536 = 131.07
A = (rounded) = 131 (value to load)

**1SA131,1DS131** ;DC2 set acceleration and deceleration commands

The values shown for velocity and acceleration must be multiplied by 65,536 (as shown) to adjust for the required integer/fraction format of the input data.

And finally, the **M**ove **R**elative (**MR**) and **M**ove **A**bsolute (**MA**) commands can be used to cause the motor to move from point to point.

Example:

1**MR**1000
1**MR**-1000
1**MA**1000

# 6.0 STEPPER OPERATION

In addition to the two axes of servo control, the **DC2** is also capable of 'simple' open loop indexing of two stepper motors. These motors are referred to as axes 3 and 4 in the **DC2** commands. The first requirement of enabling this function of the **DC2**, is to issue the **M**otor o**N** command specifying axis 3 and/or 4.

Example:

3**MN**,4**MN**

In executing this command, the **DC2** will automatically configure the first 4 digital I/O channels as outputs(refer to appendix B page 2). These channels will provide the pulse and direction signals required to interface to external stepper drivers. The channels are defined as follows:

Channel 1 (J3 pin 19) is Axis 3 Pulse
Channel 2 (J3 pin 19) is Axis 3 Direction
Channel 3 (J3 pin 19) is Axis 4 Pulse
Channel 4 (J3 pin 19) is Axis 4 Direction

The "Pulse" signals are normally high, and go low for approximatly 10 microseconds to initiate a step. The "Direction" signals are low for positive motion, and high for negative motion. Note that the digital I/O channels of the **DC2** are only capable of sinking or sourcing 1 milliamp of current, thus buffering may be required to interface these signals to external stepper drivers.

Once axes 3 and 4 are enabled with the **M**otor o**N** commands, their status and positions will appear in the replies to the **T**ell **S**tatus, **T**ell **P**osition and **T**ell **T**arget commands (the **T**ell **O**ptimal command is not supported). At the same time, certain commands will now have effect for axes 3 and 4.

For example, the pulse rate can be specified using the **S**et **V**elocity command. Similarly, the motion commands **M**ove **A**bsolute and **M**ove **R**elative will cause the appropriate number of pulses in the desired direction. Note that the **DC2** trajectory generator does not support the two stepper axes, (**no acceleration or deceleration**), pulses will be generated at a single fixed rate.

The step rate for each of the steppers can be determined by the following formula:

steps per second = 1 / (**n** * 0.00000075)

where **n** is the parameter to the Set Velocity command.

3**SV**1333,3**MR**5000 ;Will result in 5000 pulses at 1000 steps per second.

The positions of axes 3 and 4 are limited to a range from -32768 to +32767. Any motion that exceeds this range will result in position "wrap around". For example, if axis 3 is at position 32766, issueing the command 3**MR**5 will cause the position to change to -32765. For any one commanded move the stepper position register may be "wrapped around" one time. This limits the total move to no more than 65531.

Issueing the **D**efine **H**ome command to a stationery stepper axes will cause the reported position to be set to 0. Issueing the **ST**op or **AB**ort commands to the steppers while they are in motion, causes the respective axis to halt at that position (no more pulses will be generated).

Due to the limitation of the stepper position register the DC2-PC does not support velocity mode motion.

# 7.0 POINT TO POINT MOTION

To perform point to point motion, follow the steps below (examples assume axis 1):

1. Issue the **M**otor o**N** (**MN**) command to the selected axis (if it isn't already on).

Example:
1**MN**

2. Issue the **P**osition **M**ode (**PM**) command to the axis.

Example:
1**PM**

3. Issue the **S**et **V**elocity (**SV**), **S**et **A**cceleration (**SA**) and **D**eceleration **S**et (**DS**) commands for the desired motion. The default units for the parameters are:

| Trajectory command type | Parameter units |
|---|---|
| Velocity parameter | Encoder counts or steps per millisecond |
| Acceleration & Deceleration parameter | Encoder counts per millisecond per millisecond |

4. Use the **M**ove **A**bsolute (***a*MA*n***) command to start the motor moving to a position '***n***' encoder counts from the position last defined as 'home'. Use the **M**ove **R**elative (***a*MR*n***) command to start the motor moving to a position '***n***' encoder counts from its' current position. Use the **G**o **H**ome (**GH**) command to start the motor moving to its' zero position. After a command is issued, the motion will complete with no further commands from the user.

Example: 1**MA**85000
1**MR**12000
1**GH**

5. Repeat step 4 to start motions to other positions.

## 7.1 CONTINUOUS VELOCITY MOTION

To cause a motor to move at a continuous velocity until commanded to stop, follow the steps below (examples assume axis 1):

1. Issue the **M**otor o**N** (**MN**) command to the selected axis (if it isn't already on).

Example:

1**MN**

2. Issue the **V**elocity **M**ode (**VM**) command to the axis.

Example:

1**VM**

3. Issue the **S**et **V**elocity (**SV**), **S**et **A**cceleration (**SA**) and **D**eceleration **S**et (**DS**) commands for the desired motion. The default units for the parameters are:

| Trajectory command type | Parameter units |
|---|---|
| Velocity parameter | Encoder counts per millisecond |
| Acceleration & Deceleration parameter | Encoder counts per millisecond per millisecond |

Example:

1**SV**100000,1**SA**100000,1**DS**100000

4. Use the **DI**rection command to select positive or negative motion. A command parameter of 0 selects motion in the positive direction (increasing position reading), while 1 selects motion in the negative direction (decreasing position reading).

Example:

1**DI**0

5. Use the **GO** command to start the motor moving in the specified direction. The motor will accelerate at the specified rate until it reaches the specified maximum velocity.

Example:

1**GO**

6. Issue the **ST**op (**ST**) command to cause the motor to decelerate and stop. Repeat steps 4 and 5 to start motion in the opposite direction.

Example:

1**ST**

## 7.2 MULTI-AXIS CONTOURING

### Linear Interpolation

The **DC2** is also capable of linear coordinated motion of the two servo axes. On power up, the **DC2** defaults to syncronization off, in this state each axis functions independently and motion commands are executed immediatly. When syncroniation is turned on with the **S**yncronization o**N** command, the controller will accept motion commands for each axis, but motion will not be started until the **GO** command is issued. At that time, the **DC2** will start motion of both axes towards the specified targets. The respective velocities, accelerations and decelerations of the two axes will be adjusted to cause the resulting motion to occur at the set vector velocity, acceleration and deceleration. Once motion is started, new motion commands should not be issued until the axes have finished the previous move. To return the **DC2** to unsyncronized motion, the **N**o **S**yncronization command should be issued.

The vector velocity, vector acceleration and vector deceleration are set with the **VV**, **VA** and **VD** commands respectively. Note that when the **S**yncronization o**N** command is issued, the current velocities and accelerations for each axis will be saved. Any syncronized motion commands will cause these values to be changed. When the **N**o **S**yncronization command is issued, the velocities and accelerations will be restored to their previous settings.

Example:

**SN**,**VV**100000,**VA**2000,**VD**2000 <return>

Enables syncronized motion.

1**MR**5000,2**MR**-20000 <return>

Adjusts targets for both axes, but no motion occurs.

**GO** <return>

Causes both axis to start motion simultaneously. Axis 1 will move 5000 counts while axis 2 moves -20000 counts. Both axes will come to a stop at the exact same instant. Their vector velocity and acceleration/deceleration will be 100000 and 2000 respectively.

**NS** <return>

Turn syncronization off, restores velocity and accelerations of both axes.

### Circular Contouring

In applications where the **DC2** drives the perpendicular axes of an "XY" table or similar mechanism, it is capable of moving the two servos in syncronized motions that trace circular and elliptical arcs.

In order to generate a circular or elliptical arc, the **DC2** must be switched into "Arc" mode. To do this, the servos are moved to the starting point of the arc. The center of the arc is then specified in absolute coordinates using the **A**rc center **X** (**AX**, code = 180) and **A**rc center **Y** (**AY**, code = 181) commands. Axis 1 of the **DC2** is assumed to be the X axis and axis 2 the Y axis The radius of the arc is then specified using the **R**a**D**ius (**RD**, code = 182)

command for circles, or **R**adius **X** (**RX**, code = 184) and **R**adius **Y** (**RY**, code = 185) commands for ellipses. The **RX** and **RY** commands should also be used when the X and Y axes have different ratios. The final step is to specify the starting angle on the arc with the **A**n**G**le (**AG**, code = 183) command. This command will also set the status bits for both axes, placing them in arc mode.

The units for the **A**n**G**le command and the persuing arc motion commands are in units of 3.433228e-4 degrees (360/1048576). An angle of 0 is assumed to lie along the positive y axis. Positive angles are towards the positive x axis (clockwise rotation), negative angles are towards the negative x axis (counter-clockwise rotation).

With the **DC2** servos in arc mode, circular and elliptical motion can be initiated with **M**ove **A**bsolute and **M**ove **R**elative commands using 5 as the axis number. The parameter to the motion commands will be the angle of the arc using the same units as the **A**n**G**le command described above. Motions consisting of multiple rotations are allowed up to +/-2048 complete revolutions.

The speed of the servos moving around the arc can be set with the **S**et **V**elocity, **S**et **A**cceleration and **D**eceleration **S**et commands, again using 5 as the axis number. The units of the parameter to the **S**et **V**elocity command is 5.2387e-6 degrees / second. The units for the acceleration and deceleration commands are 5.2387e-3 degrees / second / second.

While the servos are moving in arc mode, their destination along the arc can be changed at any time. Their velocity and accelerations can also be changed on the fly. To stop or abort the arc motion use the **ST**op and **AB**ort commands respectively. These commands should be issued with an axis number 0 or 5.

To take the servos out of arc mode, or to change the center of the arc or its radius, the servos must stopped and changed back to position mode with the **P**osition **M**ode command. As before, 5 is used as the axis number to this command.

Examples of circular contouring can be found on the following two pages.

```
;DC2 arc demo; traverse a 270 degree arc around origin

5SV2000000,SA50000,DS50000      ;Set arc velocity and acceleration:
                                ;SV2000000 = 34 seconds for full circle.

1MA0,2MA50000,0WS100            ;Move to starting point of arc.

AX0,AY0,RD50000,AG0             ;Set center of arc, radius, and angle from
                                ;positive axis. AG sets servos in arc mode.

5MA786432,WS100,PM              ;Move absolute command to traverse arc.
                                ;786432 x 3.433228e-4 = 270 degrees.
Wait                            ;for stop, place servos in position mode.

1MA0,2MA0,0WS100                ;Move to origin (arc center).
```

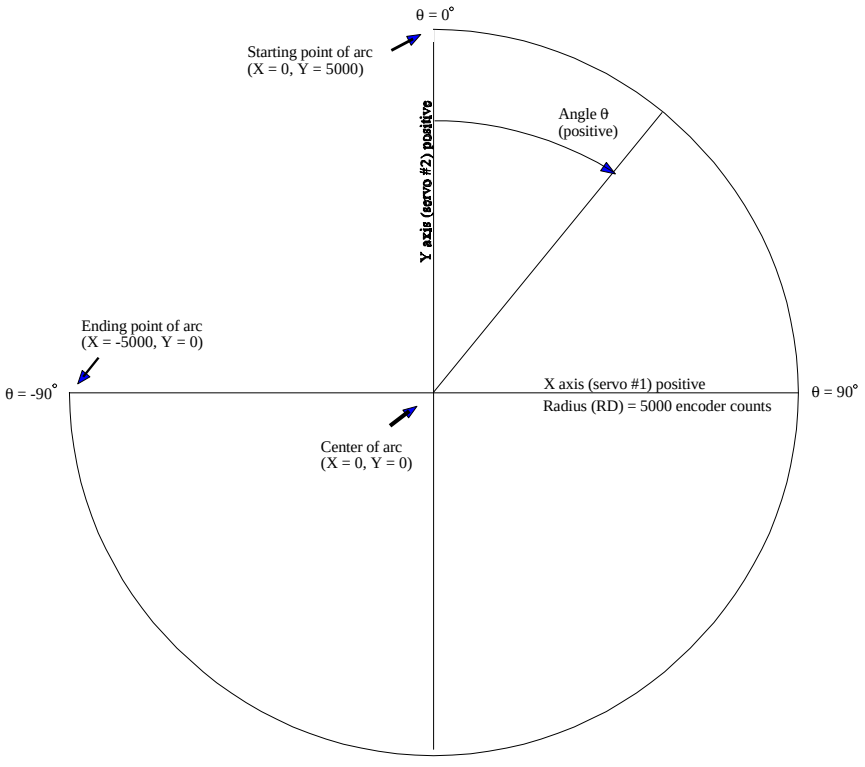


Figure 12: DC2 arc example

**Circular Contouring (cont.)**

```
;DC2 Elliptical Arc Demo; draw elliptical arc around origin
;
5SV2000000,SA50000,DS50000          ;Set arc velocity and acceleration:
                                    ; SV2000000 = 34 seconds for full circle.

1MA50000,2MA0,0WS100                ;Move to starting point of arc.

AX0,AY0,RX50000,RY25000,AG262144          ;Set center of arc, set two different
radii with                          ;the RX and RY commands, and angle from
                                    ;the positive y axis.
                                    ; 90 degrees / 3.433228e-4 = 262144. AG
                                    ;also sets servos in arc mode.

5MR-786432,WS100,PM                 ;Move relative command to traverse arc.
                                    ;-786432 x 3.433228e-4 = -270. Wait for
                                    ;stop, place servos in position mode.

1MA0,2MA0,0WS100                    ;Move to origin.
```

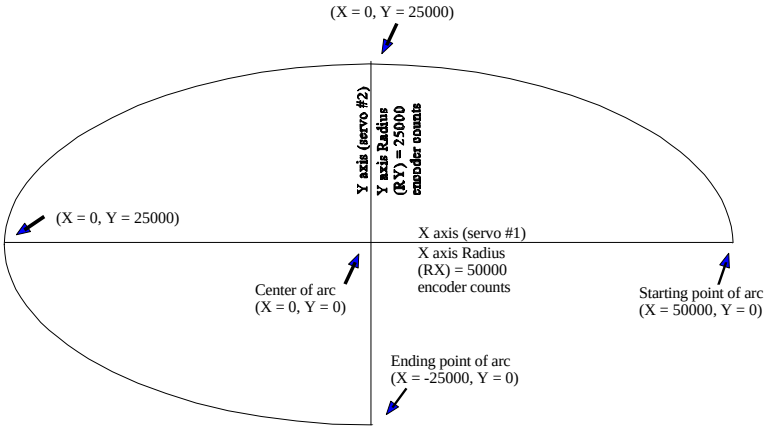


Figure 13: DC2 elipse example

**Continuous Path Contouring**

The continuous path contouring implemented on the **DC2** provides a means of driving the servos along any continuous path specified by the user. The path can consist of lines and arc segments, or can be totally random.

Executing continuous path motion on the **DC2** requires that XY points (servo positions) along a predefined path be calculated and loaded into the DC2's point memory. These calculations can be done by the host PC or other programmable controller. In order to maintain a constant tangential velocity, the XY points must be at equally spaced intervals along the path. Each XY point will consist of two servo positions in absolute encoder units. The DC2 will perform linear interpolated motion between the points in the path.

The continuous path contouring point memory holds 2048 pairs of servo positions for axes 1 and 2. It begins at address 32896 hex in the DC2's internal memory space. The following commands demonstrate how the **A**ccumulator **L**oad (**AL**) and the **W**rite **L**ong (**WL**) commands can be used to load the point memory with calculated positions:

```
AL1000,WL32896          ; Axis 1 first position = 1000
AL2000,WL32900          ; Axis 2 first position = 2000
AL1500,WL32904          ; Axis 1 second position = 1500
AL2500,WL32908          ; Axis 2 second position = 2500
etc...
```

After the first positions on the path have been stored in the contouring memory, and the servos moved to that position, the servos can be placed in **C**ontour **M**ode with the **CM** command (code=27). The parameter to this command must equal the index of the current servo positions in contouring memory. The parameter must be in the form of a 32 bit integer with an implied 16 bit fractional part. Given the example commands above, assuming the servos optimal positions are 1000 and 2000 for axes 1 and 2 respectively, the parameter to the **CM** command should be 0. No axis number is specified with the **CM** command, so the proper command string would be CM0. Alternatively, if the servos optimal positions were 1500 and 2500, then the second point (index = 1) in the contouring memory must be specified in the parameter to the **CM** command. Since this parameter must be scaled by 16 bits, the point index of 1 is multiplied by 65536 to get the proper command parameter. The command string in this case would be **CM65536**.

After the servos are in contouring mode, the **S**et **V**elocity (**SV**), **S**et **A**cceleration (**SA**) and **D**eceleration **S**et (**DS**) commands can be used to set the rate of motion along the contour path. Each of these commands should use axis 6 while in the contouring mode. The parameter to the **SV** command will be in units of 'points per millisecond'. The parameter must be in the form of a 32 bit integer with an implied decimal point to the left of the most significant bit.

For example, to move between the two points loaded in the contouring memory in one second, a value of 0.001 points per millisecond would be specified with the **SV** command. To convert this value into the proper format, multiply it by 4294967295 and discard the digits to the right of the decimal point, as shown below:

1 point per second = 0.001 points per millisecond

0.001 points per millisecond x 4294967295 = 4294967

and the command sequence would be:

```
6SV4294967                    ;Set contouring velocity
```

The **S**et **A**cceleration and **D**eceleration **S**et commands have parameters in units of 'points per millisecond per millisecond'. These parameters must also be in the form of a 32 bit integer with an implied decimal point to the left of the most significant bit. To convert an acceleration value from points per second per second, to points per millisecond per millisecond, divide it by 1000000. Then multiply it by 4294967295 and discard digits to the right of the decimal point to determine the proper command parameter.

Once the velocity, acceleration and deceleration values have been set in contouring mode, either the **M**ove Absolute or **M**ove Relative commands can be used to initiate motion along the path. Axis number 6 must be used with these commands while the servos are in contouring mode. The parameters to these commands are in units of point index. The parameters must be in the form of 32 bit integers with implied 16 bit fractional parts. As an example, to move to a position halfway between the two points loaded into the **DC2**, the target position will be index 0.5. The command parameter can be calculated by multiplying this value by 65536, and discarding the digits to the right of the decimal point, as shown below:

```
0.5 x 65536 = 32768
```

Using the **M**ove **A**bsolute command, the command string would be:

```
6MA32768                      ;Move servos to a position halfway between the first
                              ;two points in contouring memory.
```

The Tell contour indeX (TX) command can be used to indicate the location of the servos along the contour path. The value report will be the last contour point number passed.

The **ST**op and **AB**ort commands can be used to stop motion along the path while moving in continuous path contouring mode. When issuing either of these commands, use axis number 6 or don't specify an axis at all. Motion along the contour path can occur in either direction by using the appropriate move commands. The target position, velocity, acceleration or deceleration can also be changed while the servos are moving along the path.

If the **DC2** reaches the end of the point memory, it will wrap around to the beginning of the point memory. By filling the point memory ahead of the servos, a continuous motion that covers up to 32768 points can be accomplished.

## 7.3 MASTER/SLAVE

The **DC2** supports slaving one axis to the other (master axis). The slave axis is configured to run so that its' position relative to the position of the master axis maintains a constant ratio. Since the commanded position of the slave axis is based only on the current position of the master axis, it is not required that the DC2 directly control the motion of the master. This allows the DC2 to support motion of the slave axis controlled by an encoder connected to the master axis.

The Master/Slave mode is implemented by issuing the **Set Master** (***a*SM*n***) command where:

a = slave axis
n = slave axis ratio ( from 0.000000001 to 1)
where:
if ratio = 1 then ***n*** = 0
else if ratio < 0.5 then n = ratio * 4294967295
else if ratio > = 0.5 the n = (ratio * 4294967295) - 4294967295

Note: The **M**otor o**N** (***a*MN**) command and PID parameter commands must be issued set prior
to issuing the **S**lave **M**ode (***a*SM*n***) command.

Once the **S**et **M**aster command has been issued, the slave axis will begin tracking the master axis with the programmed ratio. The direction of the slave axis relative to the master axis can be set using the **DI**rection command. To terminate the master and slave connection between the axes, issue the **Set Master** command to axis 0 (***0*SM**), followed by either the **P**osition **M**ode (**PM**) or the **V**elocity **M**ode (**VM**) command.

An example of a Master/Slave configuration with the slave ratio set to 0.5 is shown below:

Master axis = axis 1, an externally controlled motor with the DC2 serving as an encoder reader only (no need to issue Motor oN or PID parameters).

Slave axis = axis 2, A motor controlled by the DC2 which is to be slaved to the encoder connected to axis 1.

```
2MN,2SG200,2SD200,SI5,DI0        ;Motor on axis 2, set PID parameters, set direction

2SM-21477483647                  ; (ratio * 4294967295) - 4294967295
                                 ; 2147483647.5 - 4294967295
                                 ; -2147483647.5
                                 ; -2147483647 (rounded)

0SM                              ;Terminate Master/Slave mode

1PM,2PM                          ;Reconfigure axis 1 & 2 for Position Mode
```

## 7.4 MANUAL POSITIONING/JOGGING

In some applications it may be necessary to have a means of manually positioning the servos. Since the **DC2** is able to control the motion of the servos with precision at both low and high speeds, all that is required is a manual command input. A joystick provides such an interface with natural hand to motion coordination.

The **DC2**, through its analog inputs, supports the use of a joystick for manual servo positioning. Typical joysticks have 2 potentiometers, one for each axis of motion. Pots with total resistances between 1K and 100K ohms are suitable for use with the **DC2**. Connect the end taps of both pots to ground and +5 VDC. The center tap for the potentiometer controlling axis #1 is connected to J3 pin 22. The center tap for the potentiometer controlling axis #2 is connected to J3 pin 20. The wiring example shown below utilizes the DC2's on board +5 vdc reference (jumper JP14 pins 2 and 3 connected) to scale the analog conversion and to supply power to the potentiometers. For additional information on the analog input interface refer to section 2.5.

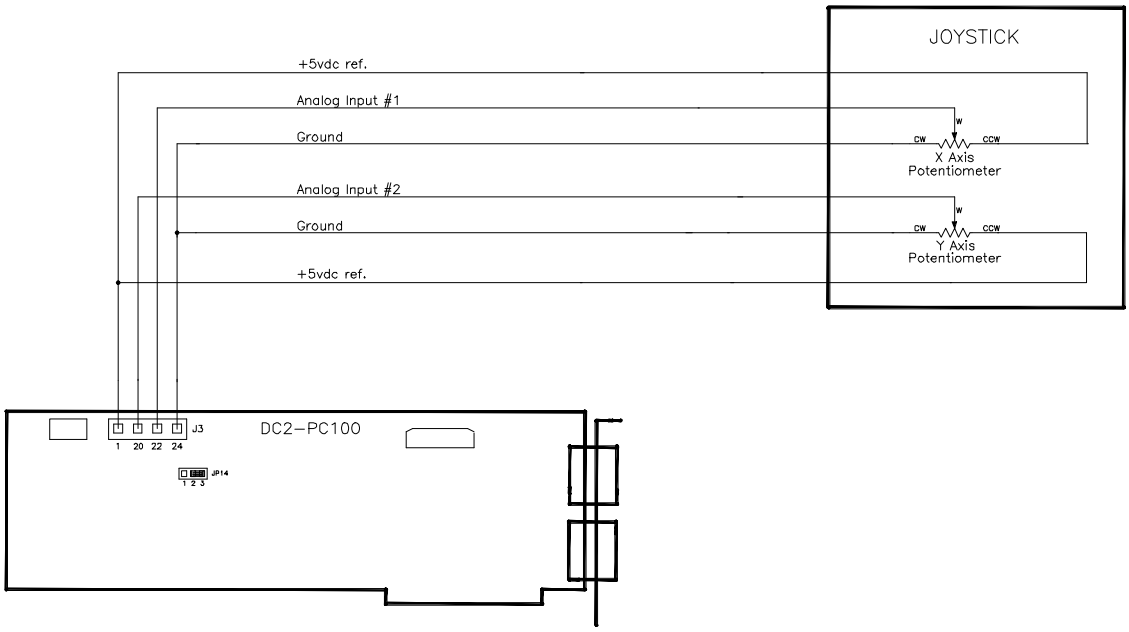


Figure 14: DC2 joystick interface

After connecting the joystick to the **DC2**, its operation can be verified with the **T**ell **A**nalog (**TA*a***) command. Since the analog inputs are measured by an 8 bit Analog to Digital Converter (ADC), the input voltage will be reported as a number between 0 and 255. With the joystick at its static center position, the pots will have a voltage of 2.5 VDC at their center taps. The Tell Analog command will report this as a reading of approximately 128 on inputs 1 and 2. Most joysticks have centering adjustments to correct offsets if necessary. Pushing the joystick to its extreme side positions, should cause the **T**ell **A**nalog command to report corresponding values between 0 and 255 for inputs 1 and 2.

Once operation of the joystick is verified, control of the servos can be initiated. The **DC2** will

use the readings from the analog inputs when the respective axis is in "Velocity" mode, and a non-zero "**J**og **G**ain" has been set. For joystick operation, set the maximum velocity to zero with the Set Velocity command, and place the servo in Velocity mode as follows:

1SV0,1VM,1GO,2SV0,2VM,2GO

At this point, the servo should be holding position, but there will be no joystick response until the **J**og **G**ain command is used to set the parameter to a non-zero value. The units of this parameter is 0.015 encoder counts per second per ADC unit. An ADC unit is 5 VDC / 256 = 0.019 Volts, this results in a **J**og **G**ain parameter with units of 0.78 encoder counts per second per volt. Given the desired maximum velocity when the joystick is pushed to the extreme, the corresponding **J**og **G**ain can be calculated by the following equation:

**J**og **G**ain = Desired Velocity / (0.78 * 2.5)

For example, if the desired maximum velocity is 1000 encoder counts per second, the Jog Gain should be set to 1000 / (0.78 * 2.5) = 513. After issuing the following command sequence the servos should move with a velocity proportional to the amount the joystick is depressed:

1JG513,2JG513

To increase or decrease the maximum velocity of the servo the **J**og **G**ain can be adjusted up or down at any time.

Depressing the joystick in the opposite direction should cause the corresponding servo to reverse direction. To change the direction that a servo moves in response to the joystick, reverse the leads on the end taps of the respective pot.

Because of mechanical and electrical variations, the joystick reading may dither around 128 when the stick returns to center. This may cause undesirable drifting of the servos when the joystick is enabled. To prevent this, another parameter called **J**og dead**B**and can be adjusted. The parameter to the **J**og **D**eadband command is in ADC units and specifies the amount that the joystick reading can very from the center before it has any effect. Once the reading exceeds the deadband level, it will increase linearly from zero. The following command sequence will provide a +/-3 ADC unit (0.06 volt) deadband in the joystick operation:

1JB3,2JB3

It was stated earlier that most joysticks have adjustments for setting the center positions. The **DC2** also has a parameter for adjusting the center position of the joystick. The **J**og **O**ffset command is used to set the center position in ADC units. This value defaults to 128 for a center voltage of 2.5 volts. Changing the **J**og **O**ffset to a different value will provide an increased velocity range in one direction, but a reduced range in the opposite direction. For example, the following command would set the center voltage for axis 1 to 2.0 volts:

1JO102

To disable the joystick, set the Jog Gains back to zero, and stop the servos with the following command sequence:

1JG0,2JG0,0ST

## 7.5 TEACHING POINTS

Points can be stored in the **DC2**'s point memory with the **L**earn **P**oint (**LP**) and **L**earn **T**arget (**LT**) commands. The **L**earn **P**oint command will store the current position into **DC2** memory. The **L**earn **T**arget command will store a commanded Target Position into **DC2** memory. The motors can then be commanded to move to those learned positions by issuing **M**ove to **P**oint commands using the point numbers as the command parameter, rather than the true positions.

Example:

**MF**
1**MA**0,2**MA**0,0**LT**1
1**MA**1000,2**MA**2000,0**LT**2
1**MA**3000,2**MA**3000,0**LT**3
1**MA**2000,2**MA**1000,0**LT**4
1**MN**,2**MN**,**AL**1
1**MP**@0,2**MP**@0,0**WS1**0,**AA**1,**RP**3

This example begins by issuing a motor off command to the **DC2**. Move commands are then issued to axes 1 and 2. No motion will occur because the motors are in the off state, but the target positions will be updated. After each simulated move, a **L**earn **T**arget command is issued with an axis specifier of 0. This will cause the respective motor target positions to be stored in the **DC2**'s memory. After teaching 4 position pairs, the motors are turned back on and a command sequence that will move axes 1 and 2 to the learned positions 1,2,3 and 4 is issued. The axes will stop for 1 tenth second at each position before continuing to the next point. The axes will move with the accelerations and velocities that were set for those axes prior to the **MP** command being issued.

There are 16384 bytes of RAM available for point storage. Each point stored uses 4 bytes of memory per axis. Since there are two servos on the DC2, the maximum number of points that can be saved is 2048.

Available bytes of RAM / bytes per point / number of servos
18364 / 4 / 2 = 2048

The DC2 can perform linear interpolated moves using learned points when syncronization is on (SN). To set the motion parameters use the VV, VA, and VD commands. The GO command must be issued after the MP command.To implement linear interpolated moves using the same points learned in the example above:

1**MN**,2**MN**,**AL**1
**SN,VV100000,VA200,VD200**
1**MP**@0,2**MP**@0,GO,0**WS1**0,**AA**1,**RP**3

## 7.6 HOMING MOTORS

When the **DC2** is turned on or reset, the servo positions are initialized to zero. If they are subsequently moved, the controller will report their positions relative to the position where they were initialized.

In many applications, there is some position of the motor that is considered 'home'. When the motor is at this position, it is desirable to have the controller report its' position as zero. With the **DC2** there are several ways to achieve this. One method is to physically move the motor to the home position, and then to issue a **D**efine **H**ome (***a*DH**) command to the axis. In automated system, there are advantages to having electro-mechanical devices in the system signal the **DC2** when a motor is at its home position, and programming the controller to find the home position on its' own. The remainder of this section describes how this can be done for servos motors.

The **DC2** is not shipped from the factory with pre-programmed homing functions. Instead, it has been designed to allow the user to define a custom homing sequence that is specific to the system requirement. Using any of the built-in motion commands as part of a homing macro, the user can write a sequence of moves to perform the necessary homing operation.

In addition to the **D**efine **H**ome command, there are two other commands that are used to set a motors position. The **F**ind **I**ndex (**FI**) command is used to zero a servo's position when its' encoder 'Index' input goes active. The **F**ind **E**dge (**FE**) command is used to zero a servo's position when its' 'Home' input goes active. Neither of these commands will start or stop any motor motions, it is up to the user to initiate motion prior to issuing the commands, and stopping motion after they complete.

Since an index pulse may occur at numerous points of a servo's travel (once per revolution in rotary encoders), a typical servo application will require a coarse home signal to "qualify" the index pulse. In this situation the following command string could be used to initialize servo 1:

1**MN**,**VM**,**SV**1000,**DI**0,**GO**,**WE**1,**ST**,**WS**10,**DI**1,**GO**,**WE**0,**FI**0,**ST**,**WS**10,**MN**,**PM,MA**0,**WS**10

Here the servo is placed in Velocity Mode and moves out of the coarse home region and then back into it (**WE** commands). As soon as the servo re-enters the coarse home range it is commanded to find the index pulse (**FI** command). At that point the current position will be captured in an internal register of the **DC2**. After the find index command completes, the servo is commanded to stop. After the wait stop command, a motor on command is issued to adjust the servos position relative to where the index mark was activated. As a final step the servo is moved back to the "home" position.

For applications where an encoder index is not available the following command string could be used to initialize motor 1:

1**MN**,**VM**,**SV**1000,**DI**0,**GO**,**WE**1,**ST**,**WS**10,**DI**1,**GO**,**FE**0,**ST**,**WS**10,**MN**,**PM,MA**0,**WS**10

Note that the motor is moved in the positive direction until the home input goes inactive (**WE** command). Next the motor is moved in the negative direction until the home input goes active (**FE** command). At soon as the home is sensed, the current position is defined to be 0. Finally the motor is stopped, and then moved back to the "home" position.

The two previously described homing command sequences can be preceded with a **M**acro **D**efine (**MD**) command to convert them into macros. For use with the **HO**me (**HO**) command, macro 1 should be defined to home motor 1, macro 2 should be defined to home macro 2. The macros can then be call directly with the **M**acro **C**all (**MC**) command, indirectly with the **HO**me command,

## 7.7 MOTION LIMITS

The **DC2** Motion Controller provides dedicated inputs for limit switches.

If a limit switch input goes active after it has been enabled by the motion **L**imits o**N** command, and the motor has been commanded to move in the direction of that switch, the Motor Error and one of the Limit Tripped Flags will be set in the motor status (***a*TS**). At the same time the motor will be turned off or stopped.

Once motion limit condition exists, the error flags will remain set until the motor is turned back on with the **MN** command. When the motor is turned back on, it can be moved out of the limit region with any of the standard motion commands.

The action that a motor takes when it encounters a motion limit can be selected with the **L**imit **M**ode command. The default action is to turn the motor off. Alternatively, the motor can be stopped abruptly, or decelerated to a stop using the current deceleration command. The **L**imit **M**ode command can also used to change the active level of a limit input. For applications that utilize the opto isolators, the default active level is between 2.5 and 12 vdc. For applications that utilize the TTL level inputs, the default active level is a TTL low (less than .7vdc). To invert the active level for a limit input, issue the Limit Mode command where the parameter *n* equals 128 plus the desired stop mode. For example, to configure axis 1 for an active high limit input that decelerate to a stop issue the following command:

```
1LM130
```

## 7.8 LOADING USER DATA

Servo and I/O data can be loaded into the accumulator (reg. 0) by using by using Read commands. To load servo data the axis specifier must be placed in front of the **RL**, **RW** or **RB** commands. To load analog input data no axis specifier is used. The table below lists the accessible variables and their locations. Note that the variables are listed as 32, 16 or 8 bit. The user should use the Read Long, Read Word and Read Byte commands for variables in each group respectively.

Example:

```
        1RL8        ;loads axis 1 target position into accumulator
        2RW46       ;loads axis 2 maximum torque into accumulator
```

32 BIT VARIABLES

| | | |
|---|---|---|
| STAT | 0 | ;MOTOR STATUS |
| ENCOD | 4 | ;ENCODER READING |
| TRGT | 8 | ;TARGET |
| TRAJ | 12 | ;TRAJECTORY GENERATOR (48 BIT) |
| TVEL | 18 | ;TRAJECTORY VELOCITY |
| MVEL | 22 | ;PROGRAMMED MAXIMUM VELOCITY |
| ACCEL | 26 | ;PROGRAMMED ACCELERATION RATE |
| DECEL | 30 | ;PROGRAMMED DECELERATION RATE |

16 BIT VARIABLES

| | | |
|---|---|---|
| KPOS | 34 | ;POSITION GAIN |
| KDER | 36 | ;DERIVATIVE GAIN |
| KINT | 38 | ;INTEGRAL GAIN |
| KVEL | 40 | ;VELOCITY GAIN |
| DTERM | 42 | ;DERIVATIVE TERM |
| INTLM | 44 | ;INTEGRATING LIMIT |
| MXTRQ | 46 | ;TORQUE LIMIT |
| FLERR | 48 | ;FOLLOWING ERROR |
| MXERR | 50 | ;MAXIMUM POSITION ERROR |
| INDEX | 52 | ;HIGH WORD OF INDEX |
| DBAND | 54 | ;POSITION DEADBAND |
| TQOUT | 56 | ;TORQUE OUTPUT |
| | 58 | ;OUTPUT DEADBAND |
| | 60 | ;TARGET DELAY |

8 BIT VARIABLES

| | | |
|---|---|---|
| SMPFQ | 62 | ;SAMPLING FREQUENCY |
| SMPCT | 64 | ;SAMPLING COUNT |
| | 4097 | ;ANALOG INPUT CHANNEL #1 |
| | 4098 | ;ANALOG INPUT CHANNEL #2 |
| | 4099 | ;ANALOG INPUT CHANNEL #3 |
| | 5000 | ;ANALOG INPUT CHANNEL #4 |

## 7.9 PC INTERRUPTS

The DC2 supports the capability to interrupt the PC host. After the Enable Interrupts (EI) command is issued with parameter n = 1, the DC2 will assert IRQ5 when a reply for the PC host or is nl longer busy executing a command. The interrupt line will be deactivated when a reply is read by the host, a new command is received from the host, or the host strobes the DC2's Attention register. The Enable Interrupt command issued with parameter n = 0 will disable host interrupts.

**Note**: The DC2 will pull IRQ5 **low** after a reset. This may interfere with other devices on the PC bus that use IRQ5 (including the MS Bus Mouse). If this is a problem, lift pin 15 of U17 on the DC2.

## 7.10 POSITION RECORDING/CAPTURING

The Recording/Capturing capabilities of the DC2 allows the user to implement both position recording and high speed synchronization to event.

### Position Recording

The DC2 can be commanded to record as many as 2048 positions. Positions are stored every millisecond (each time the servo loop is executed). The total record time is 2.048 seconds (2048 points X .001 seconds). Recorded data consists of both the current position and the optimal position. The DC2 can record positions for one or both axes. If position recording for both axes is required the maximum number of recorded points is limited to 1024 points per axis (total record time of 1.024 seconds).

The **P**osition **R**ecord (***a*PR*n***) command is used to initiate recording. Parameter *a* sets which axis will be recorded. If parameter *a* is set to zero both axes positions will be recorded. Parameter *n* sets the number of points to be recorded. Bit 14 of the servo status is set to one when a record function has started, it will be cleared when the record function has completed. To record 500 positions for axes 1 and 2:

```
0PR500
```

Recorded positions are display using the Display Recorded position (DRn) and Display recorded Optimal position (DOn) commands. The DC2 reports one position per reporting command. Parameter *n* is set to the position number to be reported. For one axis position recording parameter n = 0 to 2047. For two axes recording parameter n = 0 to 1023. Position data is stored sequencially in memory, after recording positions for both axes the position number is determined as follows:

One axis position record and display

```
1PR100          ;record position for 100 milliseconds
DR0             ;display current position at time t0
DR1             ;display current position at time t0 + 1 msec.
DR2             ;display current position at time t0 + 2 msec.
DR4             ;display current position at time t0 + 3 msec.
```

Two axes position record and display

```
1PR100          ;record position for 100 milliseconds
DR0             ;display current position of axis 1 at time t0
DR1             ;display current position of axis 2 at time t0
DR2             ;display current position of axis 1 at time t0 + 1 msec.
DR4             ;display current position of axis 2 at time t0 + 2 msec.
```

### Position Capture

With a minor PCB modification (see drawing on the next page), the DC2 can be configured to support a position capture input. After issuing the appropriate command, the current position of one or both axes will be captured when triggered by an external event (TTL low input). The three commands that support this capability are:

| | | |
|---|---|---|
| ***a*PA** | **P**osition capture **A**rm | ;Where ***a*** = axis number |
| ***a*PD** | **P**osition capture **D**isplay | ;To capture the position |
| ***a*DA** | position capture **D**isplay & **A**rm | ;for both axes ***a*** = 0. |

Issuing the Position capture Arm (**PA**) command will initiate the capture function. Within 30 usec. of the capture input changing to a TTL low, the current position will be captured and bit 2 (Position Captured) of the servo status word will be set to a one. Bit 2 of the servo status word will be cleared when the PA or PD commands are next issued. Issuing the **P**osition capture **D**isplay (PD) command will cause the DC2 to respond with the last position captured. Issuing the position capture Display & Arm (**DA**) command will cause the DC2 to first execute the the PD command followed by the PA command.

Example: When the position capture input is true capture the position and change the velocity.

```
1MR1000,1PA                          ;Move motor, arm capture
1RLO,IC2,JR-2,NO,1RL4,AR10,1SV100000 ;Jump Rel -2 creates a wait
                                     ;loop while status bit 2 is
                                     ;clear (input not true yet).
                                     ;When input true, store
                                     ;capture position in register
                                     ;10 and change velocity
```

**Modifications for position capture**

To capture the position for axes one and /or two it is necessary to add a wire from the DC2 microcontroller (U1) to the digital I/O connector (J3). Add the wire on the solder side of the board from U1 pin 17 to J3 pin 19. Please note that this configuration reduces the total available general purpose digital I/O channels from 16 to 15. The position capture function requires TTL level inputs, voltages outside the specified levels will cause serious damage to the DC2.

For other methods of interfacing the position capture inputs please consult the factory.

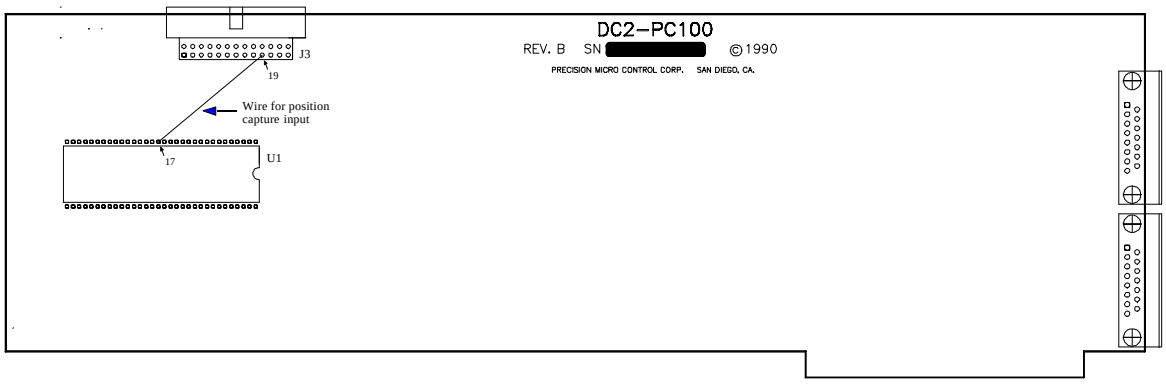


Figure 15: Position capture PCB modifications

# 8.0 DIGITAL I/O

The **DC2** has 16 undedicated digital I/O channels. These signals can be accessed on connector J3 of the motherboard. Appendix B includes a pinout for this connector.

There are numerous commands available for monitoring and controlling the digital I/O channels. Each of these commands uses a channel number from 1 to 16 as a parameter. The **C**hannel **I**n, **C**hannel ou**T** commands can be used to configure the channels individually as inputs or outputs. The **C**hannel **H**igh true logic and **C**hannel **L**ow true logic commands configures the I/O channels for "High True" or Low True" logic.

After configuration, the **T**ell **C**hannel, **C**hannel o**N**, **C**hannel of**F** commands can monitor and control the I/O channels. The sequence commands: **D**o if channel o**N**, **D**o if channel of**F**, **I**f channel o**N**, **I**f channel of**F**, **W**ait for channel o**N**, and **W**ait for channel of**F** are also available for controlling command execution.

## 8.1 ANALOG INPUTS

The **DC2** Motion Controller motherboard has 4 undedicated 8 bit analog input channels. These signals can be accessed on connector J3 of the motherboard. Appendix B includes a pinout for this connector.

The analog input channels on the motherboard are numbered from 1 to 4. Each of these inputs can accept an analog input signal from 0 to +5 volts. To prevent damage to the **DCX-PC** circuitry, the input signal must be limited to this range by external circuits.

A 5.00 volt output (from host power supply) is available on the pin 1 of connector J3. This voltage supply can be used as a reference for both an analog device (such as a potentiometer) and the Analog Reference Input (pin 23 connector J3). Alternatively, an external reference voltage between 0 and +5 volts can be connected to pin 23 of connector J3.

The **DC2** will perform a ratiometric conversion of the four input channels periodically. The result of each conversion will be a number between 0 and 255. If the on-board reference is used, the value will be the ratio of the input voltage to 5.0 volts, times 255. If an external reference voltage is used, the value will be the ratio of the input voltage to the reference voltage, times 255.

Three commands are provided on the **DC2** for reporting/reading the value of the analog inputs. The **T**ell **A**nalog command can be used to display the current readings from the analog inputs. The **G**et **A**nalog command will load a channels current reading into the accumulator (user register 0). The Read Byte command (with a parameter of 4097, 4098,4099,5000) will also load the current value of the analog channel into the accumulator, for additional information refer to section 7.9.

# 9.0 DC2 COMMAND REFERENCE

An easy-to-remember set of two letter commands has been implemented on the **DC2**. This section of the user's manual groups the commands in accordance to their purpose, describing each in detail.

Where applicable the commands are preceded by an axis number ***a*** and/or followed by a parameter valuenumber (***n***). A command that requires both will have the following form: *a***LL***n*.

For example:

1**MR**100

commands axis 1 to **M**ove **R**elative 100 (quadrature encoder) counts in the positive direction.

Please refer to appendix A for a complete listing of all commands and the associated command codes.

## 9.1 SETUP COMMANDS

### DB

set **D**ead**B**and

syntax: ***a*DB*n***

code: 118d, 76h

parameter range: 0 < = n < 2,147,483,647

explanation: Sets a range defined by parameter *n* around the servos desired position. This range is used by the 'Delay at Taget' and 'Wait for Taget' commands to define when the 'At Target flag' will be set.

### DS

**D**eceleration **S**et

syntax ***a*DS*n***

code 117d, 75h

parameter range 0 < = n < 2,147,483,647

explanation Sets the deceleration rate for a servo. The 32 bit parameter to this command is interpreted as having a 16 bit integer part and a 16 bit fractional part. This number determins how much a servo's velocity will be decreased by each millisecond while it is decelerating. The default units for the command parameter are encoder counts per second per second. When issuing contour moves; if an arc *a* = 5, if continuous path move *a* = 6. For additional information refer to sections 5.0 and 7.3.

### DT

set **D**elay at **T**arget

syntax ***a*DT*n***

code 197d, C5h

parameter range 0 < = n < 65,536

explanation This command sets the time period (in milliseconds) during which a servo must remain within the position deadband (DB) in order for the 'At Target' flag to be set. If at any the servo moves outside the deadband range, the flag well be cleared and the delay period will be restarted.

### JB

set **J**og dead**B**and

syntax ***a*JB*n***

code 223d, DFh

parameter range 0 < = n < 127

explanation This command provides compensation for mechanical and/or electrical variations when using a joystick/analog input. For a detailed description refer to section 7.5.

### JG

set **J**og **G**ain

syntax ***a*JG*n***

code 220d, DCh

parameter range 0 < = n < 512,820

explanation This command configures the desired response to a joystick/analog input. For a detailed description refer to section 7.5.

## JO

set **J**og **O**ffset

syntax ***a*JO*n***

code 222d, DEh

parameter range 0 < = n < 128

explanation This command sets the center point (no velocity command) of a joystick/analog input. The default value *n* = 128. For a detailed description refer to section 7.5.

## FR

set derivative sampling period

syntax ***a*FR*n***

code 39d, 27h

parameter range 0 < = n < 127

explanation Helps tune servo loop to the inertial characteristics of system. High inertial loads normally require a longer period and low inertial loads a shorter period. Default value is zero. For a value of *n*, the sampling period will be (*n* +1) milliseconds. For small servos this parameter can be left at the default value of 0.

## IL

set **I**ntegration **L**imit

syntax ***a*IL*n***

code 39d, 27h

parameter range 0 < *n* < 32,767

explanation Limits level of power that integral gain (SI) can use to reduce position error. This must be set to a non-zero value in order for the integral gain to have effect.

## LF

**L**imit switch of**F**

syntax ***a*LF*n***

code 54d, 36h

parameter range 0 < = n < 3

explanation Disables one or more limit switch inputs. The parameter to this command determines which inputs will be disabled. The coding of the parameter is the same as for the **L**imit switch o**N** command (**LN**). See description of **L**imit switch o**N (LN)** command for further details.

## LM

**L**imit switch **M**ode

syntax ***a*LM*n***

code 52d, 34h

parameter range 0 < = n < 130

explanation This command is used to select how the **DC2** will react when a limit switch is activated. The parameter coding is as follows:

| *n* | action |
|---|---|
| 0 | turn servo off |
| 1 | stop abruptly |
| 2 | stop smoothly |

For any of the Limit Mode options, the Motor Error flag in the status word will be set when a limit switched is sensed. This will prevent the **DC2** from moving the servo until the flag is cleared by issuing the **M**otor **O**n command. Before this command will have any effect, the limit switch must be enabled with **L**imit switch o**N** command (**LN).**

To change the active level of a limit input from TTL low to TTL high add a decimal 128 to the parameter *n*. For example, to configure axis 1 for an active high limit input that decelerate to a stop issue the following command:

example 1LM130

## LN

**L**imit switch o**N**

syntax ***a*LN*n***

code 53d, 35h

parameter range 0 < = n < 3

explanation This command is used to enable one or both of the limit switch inputs. Once enabled, if a limit switch input goes active, when the servo has been commanded to move in the direction of that switch, the servo will be stop and/or be turned off. At the same time the Limit Switch Tripped and Motor Error Flags will be set in servo status. These flags will remain set until the servo is turned back on with the **MN** command. Once the servo is turned back on, it can be moved out of the limit switch region with any of the standard motion commands. The parameter to this command determines which of the limit switch inputs will be enabled. the coding is as follows:

| *n* | Limit input enabled |
|---|---|
| 0 ,3 or not used | limit+ and limit- |
| 1 | limit+ |
| 2 | limit- |

## OD

set **O**utput **D**eadband

syntax **aOD*n***

code 195, C3h

parameter range 0 < = n < 2047

explanation This command can be used to simulate a 'frictionless servo system'. Parameter *n* modifies the analog and/or motor drive outputs to a servo. For a DC2-PC100 the parameter *n* is expressed in DAC units (0 to 2047). For a DC2-PC110 the parameter *n* is expressed in PWM units (0 to 127). The value *n* is added to a positive output and subtracted from a negative output.

## PH

set servo output **PH**ase

syntax **aPH*n***

code 115d, 73h

parameter range 0 < = n < = 1

explanation This command is used to set the servo's output phasing. The phase of the output will determine whether the **DC2** drives the servo in a direction that reduces, or increases position error. The default, or standard phasing, isequivalent to issuing this command with

a parameter of 0. The servo output can be set to reverse phase by issuing this command with a parameter of 1.

## SA

**S**et **A**cceleration

syntax **aPH*n***

code 43d, 2Bh

parameter range (0 <= *n* < 2,147,483,647)

explanation Sets the acceleration rate for a servo. The 32 bit parameter to this command is interpreted as having a 16 bit integer part and a 16 bit fractional part. This number determins how much a servo's velocity will be increased by each millisecond while it is accelerating. The default units for the command parameter are encoder counts per second per second. When issuing contour moves; if an arc *a* = 5, if continuous path move *a* = 6. For additional information refer to sections 5.0 and 7.3.

## SD

**S**et **D**erivative gain

syntax **aSD*n***

code 44d, 2Ch

parameter range 0 < = *n* < = 32,767

explanation Sets the derivative gain of a servo's PID loop. For additional information refer to sections 1.3 & 5.0.

## SE

**S**top on **E**rror

syntax **aSE*n***

code 25d, 19h

parameter range 0 < = *n* < = 32,767

explanation Sets the maximum allowable "following" error (difference between the actual position and the desired position) of a servo. If the servo is on and the following error exceeds *n*, the servo will be turned off and Motor Error flag in the servo status will be set. This flag will remain set until the servo is turned back on (MN). To disable following error checking set *n* to 32,767.

## SG

**S**et proportional **G**ain

syntax **aSG*n***

code 45d, 2Dh

parameter range 0 < = *n* < = 32,767

explanation Sets the proportional gain of a servo's PID loop. For additional information refer to sections 1.3 & 5.0

## SI

**S**et **I**ntegral gain

syntax **aSI*n***

code 46d, 2Eh

parameter range 0 < = *n* < = 32,767

explanation Sets integral gain of a servo's PID loop. The integral term accumulates a servo's static error and adjusts the output to reduce the error to zero. Note that the integration limit must be set to a nonzero value before integral gain will have any effect (see **IL** command). For additional information refer to sections 1.3 & 5.0

## SQ

**S**et tor**Q**ue

syntax **aSQ*n***

code 116d, 74h

parameter range
1 < = n < = 126 PWM motor drive (Position or Velocity mode)
1 < = n < = 2047 +/-10V Analog control (Position or Velocity mode)
-126 < = n < = 126 PWM motor drive (Torque mode)
-2047 < = n < = 2047 +/-10V Analog control (Torque mode)

explanation When an axis is configured for Position or Velocity mode, the **S**et tor**Q**ue will limit the maximum output level. An example of limiting the maximum output level of a **DC2-PC100** (analog control signanl) to 50% is shown below:

example
```
1PM              ;Axis 1 Position mode
1SQ1023          ;Limit output range to +/- 5VDC (2047 8 .5 rounded)
```

To limit the maximum output level of a **DC2-PC110** (PWM motor drive) to 50% is shown below:

example
```
1PM              ;Axis 1 Position mode
1SQ63            ;Limit output range to 50% of maximum drive current (126  * .5)
```

When an axis is in tor**Q**ue **M**ode (**QM**), the **S**et tor**Q**ue command provides the user with direct control of the output DAC. The axis will operate as a precision D/A output. Closed loop servo operation is suspended while tor**Q**ue **M**ode is enabled. The parameter ***n*** is expressed in DAC units as shown in the following table:

| Model number | Resolution | Output range | DAC unit |
|---|---|---|---|
| DC2-PC100 | 12 bit | -10V to +10V | 20V / 4096 =4.88mv per unit |
| DC2-PC110 | 8 bit | no current to max current (bipolar) | max current / 128 =___ % of max current per unit |

## SV

**S**et **V**elocity

syntax **aSV*n***

code 47d, 2Fh

parameter range 0 < = *n* < 2,147,483,647)

explanation Sets the maximum velocity (quadrature counts per millisecond) of a servo. The 32 bit parameter to this command is interpreted as having a 16 bit integer part and a 16 bit fractional part. When issuing contour moves; if an arc *a* = 5, if continuous path move *a* = 6. For additional information refer to sections 5.0 and 7.3.

example Set servo velocity to 1000 rpm using a 500 line encoder:

1000 rpm / 60 sec./min = 16.66 rps
16.66 rps x 2000 quad. counts / rev. = 33320 cps
33320 cps = 33.32 counts / millisecond
33.32 x 65536 = 2184000 (shift decimal point 16 bits)
Issue command: 1SV2184000

## VA

set **V**ector **A**cceleration

| | |
|---|---|
| syntax | **aVV*n*** |
| code | 173d, ADh |
| parameter range | 0 < = *n* < 2,147,483,647) |
| explanation | Sets vector acceleration rate for linear coordinated motion. The 32 bit parameter to this command is interpreted as having a 16 bit integer part and a 16 bit fractional part. This number determines how much the servo velocities will be increased by each millisecond while they are accelerating. The default units for the command parameter are encoder counts per second per second. For additional information refer to sections 5.0 and 7.3. |

## VD

set **V**ector **D**eceleration

| | |
|---|---|
| syntax | **aVD*n*** |
| code | 174d, AEh |
| parameter range | 0 < = *n* < 2,147,483,647 |
| explanation | Sets vector deceleration rate for linear coordinated motion. The 32 bit parameter to this command is interpreted as having a 16 bit integer part and a 16 bit fractional part. This number determines how much the servo velocities will be decreased by each millisecond while they are decelerating. The default units for the command parameter are encoder counts per second per second. For additional information refer to sections 5.0 and 7.3. |

## VG

Set **V**elocity **G**ain

| | |
|---|---|
| syntax | **aVG*n*** |
| code | 119d, 77h |
| parameter range | 0 < = *n* < = 32,767 |
| explanation | Sets velocity gain (feed forward) of a servo's PID-FF loop. For additional information refer to section 1.3. |

## VV

Set **V**ector **V**elocity

| | |
|---|---|
| syntax | **aVV*n*** |
| code | 172d, ACh |
| parameter range | 0 < = *n* < 2,147,483,647 |
| explanation | Sets maximum velocity (quadrature counts per millisecond) of servos for linear coordinated motion. The 32 bit parameter to this command is interpreted as having a 16 bit integer part and a 16 bit fractional part. The default units for the command parameter are encoder counts per second per second. For additional information refer to sections 5.0 and 7.3. |

## 9.2 MOTION COMMANDS

### AB

**AB**ort motion

syntax **aABn**

code 10d, Ah

parameter range

explanation This is an emergency stop command. The servo stops abruptly but leaves the servo loop (PID) enabled. The target position for the selected axis or axes is changed to be equal to the present position. It is used for stopping an undesired motion. When issuing contour moves; if an arc *a* = 5, if continuous path move *a* = 6.

### AG

arc **A**n**G**le

syntax **aAGn**

code 183d, B7h

parameter range (-360/ 3.433228e-4) < n < (+360/ 3.433228e-4)

explanation The arc AnGle command specifies the starting angle of the arc. The parameter *n* is specified in units of 3.433228e-4 degrees. Angle convention is the same as a compass. Positive Y axis is 0 degrees, positive X is 90 degrees, negative X is -90 degrees. For additional information refer to section 7.3.

### AX

**A**rc center axis **X**

syntax *a*AXn

code 180d, B4h

parameter range -2,147,483,647 < = *n* < = 2,147,483,647

explanation This command is used to specify the center of an arc for the X axis (axis #1). Since the arc motion is performed by two axes, this command should occur twice in a Contour Path command that initiates the arc motion. The parameter *n* specifies the center of the arc in absolute units. For additional information refer to section 7.3.

### AY

**A**rc center axis **Y**

syntax *a*AYn

code 181d, B5h

parameter range -2,147,483,647 < = *n* < = 2,147,483,647

explanation This command is used to specify the center of an arc for the Y axis (axis #2). Since the arc motion is performed by two axes, this command should occur twice in a Contour Path command that initiates the arc motion. The parameter *n* specifies the center of the arc in absolute units. For additional information refer to section 7.3.

### CM

**C**ontour **M**ode

syntax CMn

code 27d, 1Bh

parameter range 0 < = *n* < = 134217728

explanation This command configures the DC2 to implement Continuous Path Contouring. For additional information refer to section 7.3.

## DH — **D**efine **H**ome

| | |
|---|---|
| syntax | a**DH***n* |
| code | 35d, 23h |
| parameter range | -2,147,483,647 < = *n* < = 2,147,483,647 |
| explanation | Defines the current position of a servo to be *n*. From then on, all positions reported for that servo will be relative to that point. For additional information refer to section 7.7. |

## DH — **DI**rection

| | |
|---|---|
| syntax | a**DI***n* |
| code | 36d, 24h |
| parameter range | *n* = 0 for positive, *n* = 1 for negative |
| explanation | Sets the move direction of a servo when in velocity mode. Issuing this command with a parameter of 0 will cause the servo to move in a more positive direction, while a parameter of 1 will cause it to move in a more negative direction. For additional information refer to section 7.2. |

## FE — **F**ind **E**dge

| | |
|---|---|
| syntax | a**FE***n* |
| code | 11d, Bh |
| parameter range | -2,147,483,647 < = *n* < = 2,147,483,647 |
| explanation | This command is used to initialize a servo at a given position. It will remain in effect until the "coarse home" input goes active. At that time the current position of the servo will be defined to be *n*. This command will not cause any servo motion to be started or stopped. It is up to the user to initiate servo motion before issuing the command, and to stop any motion after it completes. With this command, the following command string could be used to initialize servo 1: |
| example | 1**VM**,**MN**,**SV**100000,**DI**0,**GO**,**WE**1,**ST**,**WS**100,**DI**1,**GO**,**FE**,**ST**,**WS**100,1**PM**,1**MA**0 |

Note that the servo is moved in the positive direction until the coarse home input goes inactive (**WE**1 command). Next the servo is moved in the negative direction until the coarse home input goes active (**FE** command). At soon as the coarse home is sensed, the current position is defined to be 0. Finally the servo is stopped, put into position mode, and then moved back to the "home" position. To save typing, this command sequence can be placed into a macro and executed using the **MC**, **MJ** or **HO** commands. For additional information refer to section 7.7.

## FI — **F**ind **I**ndex

| | |
|---|---|
| syntax | a**FI***n* |
| code | 12d, Ch |
| parameter range | -2,147,483,647 < = *n* < = 2,147,483,647 |
| explanation | This command is also used to initialize a servo at a given position. It will remain in effect until the "encoder index" pulse input goes active. At that time the current position of the servo will be defined to be *n*. Like the **F**ind **E**dge command, this command will not start or stop any servo motions, that is up to the user. Since the index pulse may occur at numerous points of the servos travel (once per revolution in rotary encoders), a typical application will require a coarse home signal to "qualify" the index pulse. In this situation the following command string could be used to initialize servo 1: |

example 1**VM**,**MN**,**SV**100000,**DI**0,**GO**,**WE**1,**ST**,**WS**100,**DI**1,**GO**,**WE**0,**FI**,**ST**,**WS**100,**PM**,**MA**0

Here the servo is placed in position mode and moves out of the coarse home range and then back into it (**WE** commands). As soon as the servo re-enters the coarse home range it is commanded to find the index pulse (**FI** command). At that point the current position will be set to 0. As a final step the servo is stopped, and moved back to the "home" position. For additional information refer to section 7.7.

## GH

**G**o **H**ome

syntax aGH
code 13d, Dh
parameter range
explanation Causes the specified axis or axes to move to absolute position 0. This is equivalent to a **M**ove **A**bsolute (MA) command, where the destination is 0.

## GM

**G**ain **M**ode

syntax aGM
code 8d, 8h
parameter range
explanation This command places a servo in the Gain Mode of operation. In this mode, the servo can be commanded to execute moves to specific positions. However, no velocity profile will be generated. The servo will be driven to the new target based only upon the output of the PID loop.

## GO

**GO** (start motion)

syntax aGO
code 14d, Eh
parameter range
explanation Causes one or more axes to begin motion. When in **V**elocity **M**ode (**VM**), specified axes will begin motion at constant velocity. When in **P**osition **M**ode (**PM**) with **S**yncronization o**N** (**SN**), both servo axes will begin motion to previously specified targets. For additional information refer to sections 7.2 and 7.3.

## HO

**HO**me

syntax aGO
code 15d, Fh
parameter range 1, 2
explanation This command will cause a specified "homing" macro to be executed. It is up to the user to define the macro to carry out the appropriate homing sequence for that servo (see **F**ind **E**dge and **F**ind **I**ndex commands). Issuing 1**HO** will cause macro 1 to be executed, issuing 2**HO** will cause macro 2 to be executed. Issuing this command with no servo specified will cause macro 5 to be executed. This macro can be defined to **HO**me the individual servos in some predefined sequence. For additional information refer to section 7.7.

## MA

**M**ove **A**bsolute

syntax *a***MA***n*

code 16d, 10h

parameter range -2,147,483,647 < = *n* < = 2,147,483,647

explanation Generates a motion to absolute position *n*. The absolute position is relative to the position when a home position was last defined or the board was last reset. A servo number must be specified and that servo must be in the "on" state for any motion to occur. The servo's target position will be adjusted whether the servo is on or off. When issuing contour moves; if an arc *a* = 5, if continuous path move *a* = 6. For additional information refer to sections 7.1 and 7.3.

example 1**MA**10000,2**MA**20000

## MF

**M**otor of**F**

syntax *a***MF**

code 17d, 11h

parameter range

explanation Issuing this command will place one or both servos in the "off" state. The servo outputs will go to the null level and the amplifier enable line will go inactive (low). This command can be used to prevent unwanted motion or to allow manual positioning of the unit.

## MN

**M**otor o**N**

syntax *a***MN**

code 19d, 13h

parameter range

explanation Use this command to place one or both servos in the on state (servo loop enabled). When the command is issued the target position is set to equal the current position. The amplifier enable line will go active (high), enabling an external amplifier to drive the motor.

## MR

**M**ove **R**elative

syntax *a***MN**

code 21d, 15h

parameter range -2,147,483,647 < = *n* < = 2,147,483,647

explanation This command generates a motion of *n* counts from the current position in the specified direction. A servo number must be specified and that servo must be in the "on" (MN) state for any motion to occur. The servo's target position will be adjusted whether the servo is on or off. When issuing contour moves; if an arc *a* = 5, if continuous path move *a* = 6. For additional information refer to sections 7.1 and 7.3.

example 1**MR**12345,2**MR**-87654

## NS

**N**o **S**yncronization

syntax **NS**

code 171d, ABh

parameter range

explanation Disables linear coordinated motion. Restores servo velocities and acceleration rates to their settings prior to syncronization being enabled. 6. For additional information refer to section 7.3.

## PA

**P**osition capture **A**rm

syntax ***a*PA**

code 217d, D9h

parameter range

explanation The Position capture Arm command initiates the capture function. Within 30 usec. of the capture input changing to a TTL low, the current position will be captured and bit 2 (Position Captured) of the servo status will be set to a one. Bit 2 of the servo status will be cleared when the PA command is next issued. For additional information refer to section 7.9.

## PM

**P**osition **M**ode

syntax ***a*PM**

code 23d, 17h

parameter range

explanation This command places a servo in the Position Mode of operation. In this mode, the servo can be commanded to execute moves to specific positions. The moves will be carried out using a trapezoidal velocity profile. Upon start up, or after a Reset, servos default to **P**osition **M**ode.
For additional information refer to section 7.1.

example 1**MN**,1**PM**,1**SV**10000,1**SA**100,1**MR**1000

## PR

**P**osition **R**ecord

syntax ***a*PM**

code 23d, 17h

parameter range 0 < *n* < = 2,048

explanation This command initiates position recording. Parameter *a* sets which axis will be recorded. If parameter *a* is set to zero both axes positions will be recorded. Parameter *n* sets the number of points to be recorded. Bit 14 of the servo status is set to one when a record function has started, it will be cleared when the record function has completed. For more information refer to section 7.9.

## QM

Tor**Q**ue **M**ode

syntax *a***QM**

code 9d, 9h

parameter range

explanation This command places a servo in the Torque Mode of operation. When used in conjunction with the **S**et tor**Q**ue command (refer to section 9.1) the user is given direct control of the output DAC. The axis will operate as a precision D/A output. Closed loop servo operation is suspended while tor**Q**ue **M**ode is enabled.

## RD

arc **R**a**D**ius

syntax **RD*n***

code 182d, B6h

parameter range 0 < = *n* < = 2,147,483,647

explanation This command is used to specify the radius of an arc. Since the arc motion is performed by two axes, this command should occur twice in a Contour Path command that initiates the arc motion. The parameter *n* specifies the center of the arc in absolute units. For more

information refer to section 7.3.

## RX

**R**adius of ellipse **X** axis

syntax **RX*n***

code 184d, B8h

parameter range 0 < = *n* < = 2,147,483,647

explanation When commanding the DC2 to traverse an ellipse, this command is used to specify the radius of the X axis (axis #1). Both the RX and RY commands are required to calculate the ellipse. For more information refer to section 7.3.

## RY

**R**adius of ellipse **Y** axis

syntax **RY*n***

code 185d, B9h

parameter range 0 < = *n* < = 2,147,483,647

explanation When commanding the DC2 to traverse an ellipse, this command is used to specify the radius of the Y axis (axis #2). Both the RY and RX commands are required to calculate the ellipse For more information refer to section 7.3.

## SM

**S**et **M**aster

syntax a**SM***n*

code 26d, 1Ah

parameter range see below

explanation This command causes axis *a* to be "slaved" to the other axis with a ratio defined by *n*. Before issuing this command the slave axis parameters should be initialized. The slave axis must be turned on before any motion will occur. The master axis can be moved either manually with the servo off, or under **DC2** control in either position or velocity mode, and the slave will follow. The parameter *n* has a range of 1.0 to 0.000000001. It is encoded as an unsigned 32 bit fixed point fraction, with the decimal point to the left of bit 31.

example The following formulas can be used to derive the **S**lave **M**ode command parameter value:

if ratio = 1.0 then *n* = 0
else if ratio < 0.5 then *n* = ratio * 4294967295
else if ratio >= 0.5 then *n* = (ratio * 4294967295) - 4294967295

The direction of the slave axis relative to the master can be set using the **DI**rection command. To disable slave modes for both axes, issue this command with no axis specified (axis = 0). For more information refer to section 7.4..

## SM

**S**yncronization o**N**

syntax **SN**

code 170d, AAh

parameter range

explanation Enables linear coordinated motion. Saves servo velocities and acceleration rates so that they may be restored when syncronization is disabled. After this command is issued, motion of axes will not occur until **GO** command is issued. For more information refer to section 7.3.

## ST

**ST**op

syntax *a***ST**

code 22d, 16h

parameter range

explanation Used to bring one or all servos to a halt. This command differs from the **AB**ort command in that the servos will decelerate at their preset rate instead of stopping abruptly. This command can be used to begin stopping all servos at the same time or to stop one specific servo. If an arc motion *a* = 5, if a continuous path move *a* = 6.

## VM

**V**elocity **M**ode

syntax *a***VM**

code 24d, 18h

parameter range

explanation This command places a servo in the **V**elocity **M**ode of operation. In this mode, the servo can be commanded to move in either direction with a set velocity. The servo will move in that direction until commanded to stop. When using **V**elocity **M**ode, the user must specify the direction for the servo to move using the **DI**rection (**DI**) command. After specifying the desired velocity and direction, and placing the servo in velocity mode, the servo can be started by issuing the **GO** command. While a servo is moving in **V**elocity **M**ode, the user can change the velocity by issuing new **DI**rection and **S**et **V**elocity commands. The acceleration/deceleration rate at which the servo velocity will change is determined by the **S**et **A**cceleration (**SA**) and **D**eceleration **S**et (**DS**) commands. The acceleration and decleration rates can also be changed at any time. For more information refer to section 7.2.

example

```
1VM,SV10000,SA100,DS100,DI1,GO      ;starts servo moving
1SV10100,WA100,SV10200              ;changes velocity
1ST,WS100,PM                        ;stop servo, wait for stop, position mode
```

## 9.3 REPORTING COMMANDS

### DA

position capture **D**isplay & **A**rm

syntax ***a*DA**

code 219d, DBh

parameter range

explanation Issuing the position capture Display & Arm (**DA**) command will cause the DC2 to first execute the the Position Display (PD) command followed by the Position capture Arm (PA) command. For additional information refer to section 7.9.

### DO

**D**isplay recorded **O**ptimal position

syntax ***a*DO**

code 214d, D6h

parameter range 0 < *n* < = 2,048

explanation This command reports the optimal position as recorded by the Position Record command. The recorded position reported is defined by *n* where *n* = the recorded position number. For more information refer to section 7.9.

### DR

Display Recorded position

syntax DR*n*

code 213d, D5h

parameter range 0 < *n* < = 2,048

explanation This command reports the current position as recorded by the Position Record command. The recorded position reported is defined by *n* where *n* = the recorded position number. For more information refer to section 7.9.

### HE

**HE**lp

syntax **HE**

code 72d, 48h

parameter range

explanation Lists the valid **DC2** command mnemonics for the installed software version.

### PD

**P**osition capture **D**isplay

syntax ***a*PD**

code 218d, DAh

parameter range

explanation Issuing the **P**osition capture **D**isplay (PD) command will cause the DC2 to respond with the last position captured. For additional information refer to section 7.9.

### TA

**T**ell **A**nalog to digital converter

syntax **TA***x*

code 73d, 49h

parameter range 1 < = x < = 4

explanation Reports the digitized analog input signals to the **DC2**. The four 8-bit analog input channels accessed on connectors J3 are numbered 1,2,3 and 4. For each of these channels, the **TA** command will display a number between 0 and 255. These numbers are the ratio of the analog input voltage to the reference input voltage multiplied by 256. A +5 vdc reference for the analog converter is available on the **DC2** board or can be supplied externaly on connector J3. Selecting between the two sources is done by jumper configuration. If supplying an external reference, any DC voltage between 0 and +5 vdc is acceptable. For more information refer to sections 2.5, 7.5,7.9, 8.1.

## TB

**T**ell **B**reakpoint

syntax *a***TB**

code 91d, 5Bh

parameter range

explanation Reports the current breakpoint position set with the **WP**, **WR**, **IP** or **IR** commands.

## TC

**T**ell **C**hannel

syntax **TC***x*

code 74d, 4Ah

parameter range 1 < = x < = 16

explanation Reports the on/off status on each digital I/O line. This data is reported separately for each channel. The **DC2** responds by displaying the channel number and a "1" if the channel is "on", or a "0" if the channel is "off". For more information refer to sections 2.4, 8.0.

example **TC**6,**TC**7,**TC**8

returns:
6 1
7 0
8 1

Channels 6 and 8 are on, while channel 7 is off. Note that whether a channel reports on or off when the voltage on the input line is high or low depends on the setup of the channel. When a channel has been setup as "high true" logic using the **CH** command, the **TC** command will report a "1" when the input signal is greater than 2.0 volts DC and a "0" when it is less than 0.8 volts DC. When a channel has been setup as "low true" logic using the CL command, the TC command will report a "1" when the input signal is less than 0.8 volts DC and a "0" when it is greater than 2.0 volts DC. The default setup of all digital I/O lines is for "high true" logic.

## TD

**T**ell **D**erivative gain

syntax *a***TD**

code 75d, 4Bh

parameter range

explanation Reports the last derivative gain value programmed.

## TE

**T**ell **E**rror

syntax **TE**

code 76d, 4Ch
parameter range
explanation Reports the last command interpreter error code. If no error has occured a value of 0 will be reported. Once the command is issued, the error code will be reset to 0 and the ERROR flag in the PC status register will be cleared. This command provides a means of determining the error code when using the binary mode command interface. A table listing each of the error codes in contained in appendix F.

## TF

**T**ell **F**ollowing error

syntax *a***TF**
code 77d, 4Dh
parameter range
explanation Reports following error for servos. This value reported is the difference between the trajectory generators desired position current position of the servo.

## TG

**T**ell proportional **G**ain

syntax *a***TG**
code 78d, 4Eh
parameter range
explanation Reports the last proportional gain value programmed.

## TI

**T**ell **I**ntegral gain

syntax *a***TI**
code 79d, 4Fh
parameter range
explanation Reports the last integral gain value programmed.

## TL

**T**ell integration **L**imit

syntax *a***TL**
code 80d, 50h
parameter range
explanation Reports the last integration limit value programmed.

## TO

**T**ell **O**ptimal

syntax *a***TO**
code 89d, 59h
parameter range
explanation Reports the desired position as specified by the **DC2's** trajectory generator during the last servo loop update. The difference between this value and the position reported by the **TP** command is the following error. To report the optimal position during arc motion *a* (axis) = 5, to report the optimal position during continuous path motion *a* (axis) = 6.

## TP

**T**ell **P**osition

syntax *a***TP**

code 82d, 52h

parameter range

explanation Reports the absolute position of axis *a*. It may be used to monitor motion during both **M**otor o**N** (**MN**) and **M**otor of**F** (**MF**) states. If the system is in the decimal mode, ten digits will be reported with a leading minus sign (-) if the position is less than 0. When the hex mode is in effect, eight hex digits are reported in two's complement notation. In both cases, the data is preceded by the axis number and a space, and followed by a carriage return and a line feed character.

example **TP**

returns: 1 0000000000
2 0000000000

## TQ

**T**ell tor**Q**ue

syntax ***a*TQ**

code 209d, D1h

parameter range

explanation For a board configured as a DC2-PC100, this command reports the value last sent to the Analog Output DAC (+/-2047). For a board configured as a DC2-PC110, this command reports the value last sent to the PWM channel (+/-127). This 16 bit value is stored at offset 56 in the servo tables.

## TS

**T**ell **S**tatus

syntax *a***TS**

code 83d, 53h

parameter range

explanation Reports status information related to the operation of the specified axis. The response is coded into a single 32 bit value.

The meaning of each bit is listed below:

0 = Motor On (Least Significant Bit)
1 = Motor Error
2 = Position Captured
3 = Breakpoint Reached
4 = Trajectory Complete
5 = Motor Stopping
6 = Current Direction (0 = positive, 1 = negative)
7 = Desired Direction (0 = positive, 1 = negative)
8 = At Target
9 = Phasing (0 = standard, 1 = reverse)
10 = Looking for Index
11 = Looking for Edge
12 = Servo Homed
13 = Coarse Home Input Active
14 = Position Record Enabled
15 = Synchronization On
16 = Accelerating
17 = Position Mode
18 = Velocity Mode
19 = Torque Mode
20 = Arc Mode
21 = Contour Mode
22 = Slave Mode
23 = Linear Mode
24 = Limit Mode Abort
25 = Limit Mode Stop
26 = Limit- Tripped
27 = Limit- Enabled
28 = Limit- Active
29 = Limit+ Tripped
30 = Limit+ Enabled
31 = Limit+ Active (Most Significant Bit)

## TT

**T**ell **T**arget

syntax *a***TS**

code 84d, 54h

parameter range

explanation Reports the target position. This is the absolute position to which the servo was last commanded to move to. It may be specified directly with the **M**ove **A**bsolute (**MA**) command or indirectly with **M**ove **R**elative (**MR**) command. To report the target position during arc motion *a* (axis) = 5, to report the optimal position during continuous path motion *a* (axis) = 6.

## TV

**T**ell **V**elocity

syntax *a***TV**
code 85d, 55h
parameter range
explanation Reports trajectory generator current velocity. The value reported has the same units as the parameter to the **S**et **V**elocity command. See the description of that command for further details.

## TX

Tell contour indeX

syntax **TX**
code 88d, 58h
parameter range
explanation When executing a Continuous Path Contour move this command reports the last XY point passed. For additional information refer to section 7.3.

## TZ

**T**ell index position

syntax *a***TZ**
code 90d, 5Ah
parameter range
explanation Reports the position where the index mark was found relative to the servo's position when the **DC2** was last reset or the **D**efine **H**ome command issued.

## VE

tell **VE**rsion

syntax **VE**
code 86d, 56h
parameter range
explanation Reports the revision level of the firmware in the ROM.

## 9.4 MACRO COMMANDS

### JP

**J**um**P** to command absolute

syntax **JP***n*

code 6d, 6h

parameter range 0 < = *n* < = 42

explanation Causes the DC2 to jump to the specified command in the command string. Commands are numbered consecutively, starting with 0.

### JR

**J**ump to command **R**elative

syntax **JR***n*

code 7d, 7h

parameter range -42 <= *n* <= 42

explanation Causes the DC2 to jump to the specified command relative to the current command. Commands are numbered consecutively. Jumping Relative 0 commands will cause a jump to the same command.

### MC

**M**acro **C**all

syntax **MC***n*

code 86d, 56h

parameter range 0 <= *n* <= 255

explanation This command may be used to "Call" a previously defined macro command. In contrast to the **M**acro **J**ump command, when the **DC2** enters the new macro, a record is made of where it came from. When the called macro completes, the **DC2** will execute the command immediatly following the original **MC** command. When the **MC** command is issued, if there is no macro defined by the number *n*, an error will be reported. **M**acro **C**alls can be used any place in a command string or macro command. A macro command can call another macro command, which in turn can call another macro command, and so on. The only limitation is that a macro should not call itself (ie. recursion), unless commands have been included to limit the number of times it will do so.

### MD

**M**acro **D**efinition

syntax **MD***n*

code 3d, 3h

parameter range 0 <= *n* <= 255

explanation Used to define a new macro. Any duplication of numbers will simply result in the loss of the previously defined macro using that number.

example **MD**4,**TT**,**TP**
**MD**4,1**MR**10000

This will define macro #4 twice, but only the most recent macro definition is retained. Note that when a macro is redefined, the memory containing the previous version of the command is not reused by the **DC2**. This lost memory will not be available to the **DC2** until the **R**eset **M**acros command is issued.

### MJ

**M**acro **J**ump

| | |
|---|---|
| syntax | **MJ***n* |
| code | 5d, 5h |
| parameter range | 0 <= *n* <= 255 |
| explanation | This command may be used to "Jump" to a previously defined macro command. Once the **DC2** begins executing the new macro, it has no record of how it got there. This means that the commands that appear after the **MJ** command will not be executed. If there is no macro defined by the number *n*, an error will be reported. The **M**acro **J**ump command can be used any place in a command string or macro command. It is also acceptable for a macro to jump to itself. This is an easy way to create polling or execution loops. |

## RM

**R**eset **M**acros

| | |
|---|---|
| syntax | RM |
| code | 4d, 4h |
| parameter range | |
| explanation | Used to initialize the memory space reserved for macro commands. It is the only way in which macro commands may be removed after they are defined. This method removes all macro commands currently defined, so those macro commands still desired must be reprogrammed. It is always a good idea to use the **R**eset **M**acro command (**RM**) before setting up a new set of macro commands. |

## TM

**T**ell **M**acros

| | |
|---|---|
| syntax | **TM***n* |
| code | 81d, 51h |
| parameter range | -1 <= *n* <= 255 |
| explanation | Displays the commands which make up any macros which have been defined. If n = -1, all macros will be displayed. The **TM** command is useful for confirming the existence and/or contents of macro commands. |

## 9.5 CHANNEL COMMANDS

### CF

**C**hannel of**F**

syntax **CF***x*

code 31d, 1Fh

parameter range 1 < = *x* < = 16

explanation Causes digital I/O channel *x* to go to "off" state. If the channel has been configured for "high true", the channel will be at a logic low (less than 0.4 volts DC) after this command is executed. If it has been configured for "low true", the channel will be at a logic high (greater than 2.4 volts DC). This command requires a channel number to have any effect.

### CH

**C**hannel **H**igh true logic

syntax **CH***x*

code 66d, 42h

parameter range 1 < = *x* < = 16

explanation Causes digital I/O channel *x* to be configured for "high true" logic. This means that the I/O channel will be at a high logic level (greater than 2.4 volts DC) when the channel is "on", and at low logic level (less than 0.4 volts DC) when the channel is "off". Note that issuing this command will not cause the I/O channel to change its current state. Issuing this command without specifying a channel will cause all channels present on the **DC2** to be configured as "high true".

### CI

**C**hannel **I**n

syntax **CI***x*

code 32d, 20h

parameter range 1 < = *x* < = 16

explanation Used to configure digital I/O channel *x* as an input. All digital channels on the **DC2** default to inputs on reset or power up. This command only needs to be issued if the channel was previously changed to an output. The state of the channel can be determined with the **T**ell **C**hannel command. This command requires a channel number to have any effect.

### CL

**C**hannel **L**ow true logic

syntax **CL***x*

code 67, 43h

parameter range 1 < = *x* < = 16

explanation Causes digital I/O channel *x* to be configured for "low true" logic. This means that the I/O channel will be at a low logic level (less than 0.4 volts DC) when the channel is "on", and at high logic level (greater than 2.4 volts DC) when the channel is "off". Note that issuing this command will not cause the I/O channel to change its current state. Issuing this command without specifying a channel will cause all channels present on the **DC2** to be configured as "low true".

### CN

**C**hannel o**N**

syntax **CN***x*

code 33, 21h

parameter range 1 < = *x* < = 16

explanation Causes channel *x* to go to "on" state. If the channel has been configured for "high true", the channel will be at a logic high (greater than 2.4 volts DC) after this command is executed. If it has been configured for "low true", the channel will be at a logic low (less that 0.4 volts DC). This command requires a channel number to have any effect.

## CT

**C**hannel ou**T**

syntax **CT***x*

code 34d, 22h

parameter range 1 < = *x* < = 16

explanation Used to configure digital I/O channel *x* as an output. The **DC2** will turn the channel "off" before changing it to an output. The state of the channel can be determined with the **T**ell **C**hannel command even if it has been turned into an output. This command requires a channel number to have any effect.

## 9.6 SEQUENCE COMMANDS

### DF

**D**o if channel of**F**

syntax **DF***x*
code 107d, 6Bh
parameter range 1 < = *x* < = 16
explanation Used for conditional execution of commands. If the specified digital I/O channel is "off", commands that follow on the command line or in the macro will be executed. Otherwise the rest of the command line or macro will be skipped.

### DN

**D**o if channel o**N**

syntax **DN***x*
code 106d, 6Ah
parameter range 1 < = *x* < = 16
explanation Used for conditional execution of commands. If the specified digital I/O channel is "on", commands that follow on the command line or in the macro will be executed. Otherwise the rest of the command line or macro will be skipped.
example **DN**2,1**MR**1000 ;If channel 2 is on move 1000, otherwise no move is made.

### IB

**I**f accumulator is **B**elow

syntax **IB***n*
code 165d, A5h
parameter range
explanation **I**f accumulator is **B**elow (less than) constant *n*, do next command, else skip 2 commands

### IC

**I**f **C**lear

syntax **IC***n*
code 161, A1h
parameter range 1 < = *x* < = 32
explanation **I**f bit *n* of accumulator is **C**lear, do next command, else skip 2 commands

### IE

**I**f **E**qual

syntax **IE***n*
code 162d, A2h
parameter range -32767 < = *n* < = 32767
explanation If accumulator **E**quals constant *n*, do next command, else skip 2 commands

### IF

**I**f channel of**F**

syntax **IF***x*
code 109d, 6Dh
parameter range 1 < = *x* < = 16
explanation **I**f channel of**F**, do next command, else skip two 2 commands. Used for conditional

execution of commands. If specified digital I/O channel is "off", commands that follow on the command line or in the macro will be executed. Otherwise the next two commands of the command line or macro will be skipped. See also **N**o **O**peration (**NO**) and **B**rea**K** (**BK**) commands.

## IG

**I**f accumulator is **G**reater

syntax **IG***n*
code 164d, A4h
parameter range -32767 < = *n* < = 32767
explanation **I**f accumulator is **G**reater than constant *n*, do next command, else skip next 2 commands

## IN

**I**f channel o**N**

syntax **IN***x*
code 108, 6Ch
parameter range 1 < = *x* < = 32
explanation **I**f channel o**N**, do next command, else skip two commands. Used for conditional execution of commands. If specified digital I/O channel is "on", commands that follow on the command line or in the macro will be executed. Otherwise the next two commands of the command line or macro will be skipped. See also **N**o **O**peration (**NO**) and **B**rea**K** (**BK**) commands.

example

```
IN3,MC1,BK,MC2          ;If channel 3 on then execute macro 1, else execute
                        ;macro 2.
1SV1000,IN4,1SV100,NO,1MR500  ;Set velocity to 1000, if channel 4 on,lower velocity
                        ;to 100, move 500. Note use of NO command to
fill                    ;command space.
```

## IP

**I**nterrupt on absolute **P**osition

syntax *a***IP***n*
code 96d, 60h
parameter range -32767 < = *n* < = 32767
explanation This command is used to indicate when an axis has reached a specific position. The position is relative to the point defined as home. When the specified position has been reached, the **DC2** will set the "breakpoint reached" flag in the servo status for that axis. This command will not cause the DC2 to interrupt the PC host. The **IP** command can be issued to an axis before or after it has been commanded to move. See also, commands **IR**, **WP** and **WR**.

## IR

**I**nterrupt on **R**elative position

syntax *a***IR***n*
code 97d, 61h
parameter range -32767 < = *n* < = 32767
explanation This command is used to indicate when an axis has reached a specific position. The position is relative to the target position when a move command is issued. When the specified position has been reached, the **DC2** will set the "breakpoint reached" flag in the servo status for that axis. This command will not cause the DC2 to interrupt the PC host. The **IR** command can be issued to an axis before or after it has been commanded to move. See also, commands **IP**, **WP** and **WR**.

## IS

**I**f bit **S**et

syntax **IS***n*
code 160d, A0h
parameter range 1 < = *x* < = 32
explanation **I**f bit *n* of accumulator **S**et, do next command, else skip 2 commands

## IU

**I**f accumulator is **U**nequal

syntax **IU***n*
code 163d, A3h
parameter range -32767 < = *n* < = 32767
explanation **I**f accumulator is **U**nequal to constant *n*, do next command, else skip next 2 commands

## RP

**R**e**P**eat

syntax **RP***n*
code 100d, 64h
parameter range 0 <= *n* <= 65,535
explanation This command causes the command string to repeat *n* more times. If *n* is not specified or is 0 then commands are repeated indefinitely.

example 1**TP**,**RP**999 ;Displays the position of axis 1, 1000 times.

## WA

**WA**it

syntax **WA***n*
code 101d, 65h
parameter range 1 <= *n* <= 65,535
explanation Insert a wait period of *n* millisecond before going on to the next command.

example 1**TP**,**WA**1000,**RP**9 ;will display the position of axis 1, 10 times with a ;delay of approximately one second between ;position reports.

## WE

**W**ait for **E**dge

syntax *a***WE***x*
code 102d, 66h
parameter range 1 <= *n* <= 65,535
explanation Wait until the "coarse home" input on an axis is at the specified logic level, and then continue operation. If *x* is not specified or is 0, wait for coarse home to go active. If *x* is 1, wait for coarse home to go inactive.

## WF

**W**ait for channel of**F**

syntax **WF***x*
code 103d, 67h
parameter range 1 <= *n* <= 65,535
explanation Wait until digital I/O channel *x* is "off" before continuing to next command on command line or in macro.

## WN

**W**ait for channel o**N**

syntax **WN***x*
code 104d, 68h
parameter range 1 <= *n* <= 65,535
explanation Wait until digital I/O channel *x* is "on" before continuing to next command on command line or in macro.

## WP

**W**ait for **A**bsolute position

syntax *a***WP***n*
code 98d, 62h
parameter range -32767< = *n* < = 32767
explanation This command works the same as the **I**nterrupt on absolute **P**osition command except that the command remains in effect until the specified position is reached. See the description of the **IP** command for further details.

## WR

**W**ait for **R**elative position

syntax *a***WR***n*
code 99d, 63h
parameter range 1 <= *n* <= 65,535
explanation This command works the same as the **I**nterrupt on **R**elative position command except that the command remains in effect until the specified position is reached. See the description of the **IR** command for further details.

## WS

**W**ait for **S**top

syntax *a***WS***n*
code 105d, 69h
parameter range 0 <= *n* <= 65535
explanation Wait for *n* milliseconds after the trajectory complete flag (servo status bit 4) of axis *a* (*a* = 0 means all axes) has been set. Actual motion complete is dependent on following error and PID settling. Used for synchronizing motion and for specifying complex motions.

example
```
1MR1000,WS120,MR-1000      ;Move relative 1000 counts, wait 120 msec. After
                           ;trajectory complete bit set, move relative -1000
                           ;counts.
```

If the **W**ait **S**top (**WS**) command was not used in the example, there would be no motion of the axis. The reason being that the target position would simply be changed twice. The computer would add 1000 counts to the target position then subtract the same amount. This would take place far quicker than the axis could begin moving.

## WT

Wait for Target

syntax aWT
code 98d, 62h
parameter range
explanation This command causes the DC2 to suspend command execution until the 'AT Target' flag in the servo status word is set. Refer to 'DB' and 'DT' commands in section 9.1 .

## 9.7 LEARN COMMANDS

### LP

**L**earn **P**osition

syntax *a***LP***n*

code 112d, 70h

parameter range 0 <= *n* <= 2047

explanation Used for storing the coordinates of the current position for one or both servos in a dedicated point memory. The servos can then be moved to those points with the **MP** command. If it is desired that the position for only one axis be taught, the axis number must be specified in the command. Bear in mind that each point stores a position for each axis.

Any number of points may be stored, up to the capacity of available memory. There are 16384 bytes of RAM available for point storage. Each point stored uses 4 bytes of memory per axis. Since there are two servos on the DC2, the number of points that can be saved is 2048 (16384 / 4 / 2).

### LT

**L**earn **T**arget

syntax *a***LT***n*

code 113d, 71h

parameter range 0 <= *n* <= 2047

explanation Used for storing the coordinates of the target position for one or both servos in a dedicated point memory. If it is desired that the target for only one axis be taught, the axis number must be specified in the command. This command is similar to the **LP** command, but does not require actually moving to the position to store the point. One use of this command, is to download coordinates from a host computer or CAD system. To do this, simply turn off the servos with the **MF** command, and teach the points by issuing commands in the following sequence:

example
```
1MA10000,2MA20000,LT1,1MA20000,2MA40000,LT2      ;Teach Points
MP1,WS100,MP2,WS100                              ;Move to taught points
```

### MP

**M**ove to **P**oint

syntax *a***MP***n*

code 20d, 14h

parameter range 0 <= *n* <= 2047

explanation Used for moving to a previously stored set of coordinates. The point to be moved to is specified by *n*. If it is desired that only one axis move, you must specify which axis in the command. Points are stored with the **L**earn **P**osition (**LP**) and **L**earn **T**arget (**LT**) commands.

example
```
MP4,0WS100,MP5,0WS100,MP6,0WS100    ;will move to points 4, 5, &  6,
                                    ;stopping for 100 milliseconds at
                                    ;each point.
```

## 9.8 REGISTER COMMANDS

The DC2 contains 256 general purpose registers. These registers can be used for many purposes, including holding command parameters, performing math computations and controlling command execution. Each register is 32 bits and can hold a number between 2147483647 and -2147483648.

The registers are numbered 0 through 255, with register 0 being the 'accumulator'. The accumulator (register 0) is used in most commands that manipulate the registers.

The user can also issue commands, specifying a register number as the parameter instead of a constant. This is done by placing an '@' sign in front of the command parameter. For example, the command "1**MR**@10" will cause the **DC2** to move axis 1 by the number stored in register 10. The use of a register specifier can be used in any command as the parameter. The **DC2** does NOT support the use of the '@' sign in front of an axis number.

### AA

**A**ccumulator **A**dd

| | |
|---|---|
| syntax | **AA***n* |
| code | 133d, 85h |
| parameter range | ACC = ACC + *n* |
| explanation | |

### AC

**A**ccumulator **C**omplement, bit wise

| | |
|---|---|
| syntax | **AC***n* |
| code | 140d, 8Ch |
| parameter range | ACC = !ACC |
| explanation | |

### AD

**A**ccumulator **D**ivide

| | |
|---|---|
| syntax | **AD***n* |
| code | 136d, 88h |
| parameter range | (R0 = R0,R1 / *n* ; R1 = remainder) |
| explanation | Performs 64 bit by 32 bit signed division with 32 bit result and 32 bit remainder. Low order 32 bits of numerator must be in Accumulator (Reg. 0), high order 32 bits must be in Reg. 1. Result will be returned in Accumulator (Reg. 0), remainder in Reg. 1. |

### AE

**A**ccumulator **E**xclusive or with *n*, bit wise

| | |
|---|---|
| syntax | **AE***n* |
| code | 143d, 8Fh |
| parameter range | (ACC = ACC eor *n*) |
| explanation | |

### AL

**A**ccumulator **L**oad with constant *n*

| | |
|---|---|
| syntax | **AL***n* |
| code | 130d, 82h |

parameter range (ACC = *n*)
explanation

## AM

**A**ccumulator multiply

syntax **AM***n*
code 135d, 87h
parameter range (R0,R1 = R0 **n*)
explanation Performs 32 bit by 32 bit signed multiply with 64 bit result. Low order 32 bits of result in accumulator (Reg. 0), high order 32 bits in Reg. 1.

## AN

**A**ccumulator a**N**d with *n*, bit wise

syntax **AN***n*
code 141d, 8Dh
parameter range ACC = ACC and *n*
explanation

## AO

**A**ccumulator **O**r with *n*, bit wise

syntax **AO***n*
code 142d, 8Eh
parameter range ACC = ACC or *n*
explanation

## AR

copy **A**ccumulator to **R**egister *n*

syntax **AR***n*
code 132d, 84h
parameter range REG*n* = ACC
explanation

## AS

**A**ccumulator **S**ubtract

syntax **AS***n*
code 134d, 86h
parameter range ACC = ACC - *n*
explanation

## RA

copy **R**egister *n* to **A**ccumulator

syntax **RA***n*
code 131d, 83h
parameter range ACC = REG*n*
explanation

## RB

**R**ead **B**yte (8 bit value) at absolute memory location *n* into accumulator

syntax RB*n*
code 150d, 96h

parameter range ACC = *n*
explanation

## RL

**R**ead **L**ong (32 bit value) at absolute memory location *n* into accumulator

syntax **RL***n*
code 152d, 98h
parameter range ACC = *n*
explanation

## RW

**R**ead **W**ord (16 bit value) at absolute memory location *n* into accumulator

syntax **RW***n*
code 151d, 97h
parameter range ACC = (*n*)
explanation

## SL

**S**hift accumulator **L**eft *n* bits

syntax **SL***n*
code 144d, 90h
parameter range ACC = ACC << *n*
explanation Low order bits will be filled with zero.

## SR

**S**hift **R**ight accumulator *n* bits

syntax **SR***n*
code 145d, 91h
parameter range ACC = ACC >> *n*
explanation High order bits will be filled with zero.

## TR

**T**ell contents of **R**egister *n*

syntax **TR***n*
code 150d, 96h
parameter range 0 < = *n* < = 256
explanation Issuing the TR command with no specified parameter will report the contents of the accumulator

## WB

**W**rite accumulator low **B**yte (8 bits) to absolute memory location *n*

syntax **WB***n*
code 153d, 99h
parameter range (*n*) = ACC
explanation

## WL

**W**rite accumulator **L**ong (32 bits) to absolute memory location *n*

syntax **WL***n*
code 155d, 9Bh

parameter range (*n*) = ACC
explanation

## WW

**W**rite accumulator low **W**ord (16 bits) to absolute memory location *n*

syntax WW*n*
code 154d, 9Ah
parameter range (*n*) = ACC
explanation

## 9.9 MISCELLANEOUS COMMANDS

### BK

**B**rea**K**

syntax **BK**
code 121d, 79h
parameter range
explanation Execution of this command will cause the rest of the command line or macro to be skipped. This command is used in conjunction with the sequence commands to implement conditional execution.

### BM

**B**inary **M**ode

syntax **BM**
code 59d, 3Bh
parameter range
explanation Input and Output commands and numbers in binary format.

### BR

**B**aud **R**ate

syntax **BR***n*
code 30d, 1Eh
parameter range See the table below
explanation Used to change the programmed baud rate for the RS-232 serial communications interface. The actual baud rate is determined by using the value *n* in a countdown circuit.

example **BR**725 ;sets the baud rate to 2,400 baud

Note: This command takes effect immediately, so use with caution. Any value entered will result in some baud rate but not necessarily a standard one.

Parameter *n* codes (decimal) for the standard baud rates:

| Baud Rate | Parameter *n* |
|---|---|
| 19200 | 107 |
| 9600 | 213 |
| 4800 | 469 |
| 2400 | 725 |
| 1200 | 981 |
| 600 | 1237 |
| 300 | 1493 |

### DM

**D**ecimal **M**ode

| | |
|---|---|
| syntax | **DM** |
| code | 130d, 82h |
| parameter range | Input and output numbers in decimal format. |
| example | **DM**,**SV**12700 |

## EF

**E**cho of**F**

| | |
|---|---|
| syntax | **EF** |
| code | 37d, 25h |
| parameter range | Causes the **DC2** not to echo characters received through the PC or RS-232 serial communications interface. Only data specifically requested from the **DC2** will be transmitted. Normally used when operating the host or terminal in half duplex mode. |

## EI

**E**nable Interrupts

| | |
|---|---|
| syntax | **EI*n*** |
| code | 124d, 7Ch |
| parameter range | 0 < = n < = 1 |
| explanation | This command, when issued with parameter n = 1, will assert IRQ5 when the DC2 has a reply ready or is ready for the next command. The interrupt line will be deactivated when a reply is received from the host, a new command is received, or the Attention register is strobed. The Enable Interrupt command issued with parameter n = 0 will disable this feature.<br><br>**Note**: The DC2 will pull IRQ5 **low** after a reset. This may interfere with other devices on the PC bus that use IRQ5 (including the MS Bus Mouse). If this is a problem, lift pin 15 of U17 on the DC2. |

## EN

**E**cho o**N**

| | |
|---|---|
| syntax | **EN** |
| code | 38d, 26h |
| parameter range | |
| explanation | Causes all characters received through the PC or RS-232 serial communications interface to be echoed as they are received. Normally used when operating with the host or terminal in full duplex mode. |

## HF

**H**andshake of**F**

| | |
|---|---|
| syntax | **HF** |
| code | 48d, 30h |
| parameter range | |
| explanation | Disables hardware handshake of RS-232 serial communications interface. |

## HM

**H**exadecimal **M**ode

| | |
|---|---|
| syntax | **HM** |
| code | 61d, 3Dh |
| parameter range | |
| explanation | Input and output in hexadecimal format. |

example: **HM**,**SV**7FFF

## HN

**H**andshake o**N**

| | |
|---|---|
| syntax | **HN** |
| code | 49d, 31h |
| parameter range | |
| explanation | Enables hardware handshake of RS-232 serial communications interface. |

## NO

**N**o **O**peration

| | |
|---|---|
| syntax | **NO** |
| code | 120d, 78h |
| parameter range | |
| explanation | This command does nothing. It can be used to cause short delays in command line executions or as a filler in sequence commands. |

## RT

**R**ese**T** DC2

| | |
|---|---|
| syntax | **RT** |
| code | 42d, 2Ah |
| parameter range | |
| explanation | Performs a complete restart of the DC2 board, including restoration of all default conditions, such as acceleration and velocity, and leaves all axes in the "off" state. |

## XN

set **X**o**N** code

| | |
|---|---|
| syntax | **XN***n* |
| code | 126d, 7Eh |
| parameter range | 0 <= *n* <= 255 |
| explanation | This command sets the 8-bit code that will be sent out the serial interface when a command has completed. The XOFF codes for the serial communications interfaces default to 62. This causes a ">" prompt character to be sent when the DC2 has completed the previous command. The standard **XON** code is 17 decimal. One use of this command is to sound the bell when each command has completed. |
| example | **XN**7 ;Sounds bell when previous command finished |

# APPENDIX A: COMMAND CODE SUMMARY

| Decimal | Binary | Mnemonic |
|---|---|---|
| 2 | 2 | MC |
| 3 | 3 | MD |
| 4 | 4 | RM |
| 5 | 5 | MJ |
| 6 | 6 | JP |
| 7 | 7 | JR |
| 8 | 8 | GM |
| 9 | 9 | QM |
| 10 | A | AB |
| 11 | B | FE |
| 12 | C | FI |
| 13 | D | GH |
| 14 | E | GO |
| 15 | F | HO |
| 16 | 10 | MA |
| 17 | 11 | MF |
| 19 | 13 | MN |
| 20 | 14 | MP |
| 21 | 15 | MR |
| 22 | 16 | ST |
| 23 | 17 | PM |
| 24 | 18 | VM |
| 25 | 19 | SE |
| 26 | 1A | SM |
| 27 | 1B | CM |
| 30 | 1E | BR |
| 31 | 1F | CF |
| 32 | 20 | CI |
| 33 | 21 | CN |
| 34 | 22 | CT |
| 35 | 23 | DH |
| 36 | 24 | DI |
| 37 | 25 | EF |
| 38 | 26 | EN |
| 39 | 27 | FR |
| 40 | 28 | IL |
| 42 | 2A | RT |
| 43 | 2B | SA |
| 44 | 2C | SD |
| 45 | 2D | SG |
| 46 | 2E | SI |
| 47 | 2F | SV |
| 49 | 31 | HN |
| 52 | 34 | LM |
| 53 | 35 | LN |
| 54 | 36 | LF |
| 59 | 3B | BM |
| 60 | 3C | DM |
| 61 | 3D | HM |
| 66 | 42 | CH |
| 67 | 43 | CL |
| 72 | 48 | HE |
| 73 | 49 | TA |
| 74 | 4A | TC |
| 75 | 4B | TD |
| 76 | 4C | TE |
| 77 | 4D | TF |
| 78 | 4E | TG |
| 79 | 4F | TI |
| 80 | 50 | TL |
| 81 | 51 | TM |
| 82 | 52 | TP |
| 83 | 53 | TS |
| 84 | 54 | TT |
| 85 | 55 | TV |
| 86 | 56 | VE |
| 87 | 57 | TR |
| 88 | 58 | TX |
| 89 | 59 | TO |
| 90 | 5A | TZ |
| 91 | 5B | TB |
| 96 | 60 | IP |
| 97 | 61 | IR |
| 98 | 62 | WP |
| 99 | 63 | WR |
| 100 | 64 | RP |
| 101 | 65 | WA |
| 102 | 66 | WE |
| 103 | 67 | WF |
| 104 | 68 | WN |
| 105 | 69 | WS |
| 106 | 6A | DN |
| 107 | 6B | DF |
| 108 | 6C | IN |
| 109 | 6D | IF |
| 112 | 70 | LP |
| 113 | 71 | LT |
| 115 | 73 | PH |
| 116 | 74 | SQ |
| 117 | 75 | DS |
| 118 | 76 | DB |
| 119 | 77 | VG |
| 120 | 78 | NO |
| 121 | 79 | BK |
| 124 | 7C | EI |
| 125 | 7D | XN |
| 126 | 7E | XF |
| 130 | 82 | AL |
| 131 | 83 | RA |
| 132 | 84 | AR |
| 133 | 85 | AA |
| 134 | 86 | AS |
| 135 | 87 | AM |
| 136 | 88 | AD |
| 140 | 8C | AC |
| 141 | 8D | AN |
| 142 | 8E | AO |
| 143 | 8F | AE |
| 144 | 90 | SL |
| 145 | 91 | SR |
| 150 | 96 | RB |
| 151 | 97 | RW |
| 152 | 98 | RL |
| 153 | 99 | WB |
| 154 | 9A | WW |
| 155 | 9B | WL |
| 160 | A0 | IS |
| 161 | A1 | IC |
| 162 | A2 | IE |
| 163 | A3 | IU |
| 164 | A4 | IG |
| 165 | A5 | IB |
| 170 | AA | SN |
| 171 | AB | NS |
| 172 | AC | VV |
| 173 | AD | VA |
| 174 | AE | VD |
| 180 | B4 | AX |
| 181 | B5 | AY |
| 182 | B6 | RD |
| 183 | B7 | AG |
| 184 | B8 | RX |
| 185 | B9 | RY |
| 195 | C3 | OD |
| 197 | C5 | DT |
| 198 | C6 | WT |
| 209 | D1 | TQ |
| 212 | D4 | PR |
| 213 | D5 | DR |
| 214 | D6 | DO |
| 217 | D9 | PA |
| 218 | DA | PD |
| 219 | DB | DA |
| 220 | DC | JG |
| 222 | DE | JO |
| 223 | DF | JB |

# APPENDIX B: DC2 CONNECTORS AND JUMPERS

(Refer to diagram at end of Appendix B)

**J1: DC SERVO CONNECTOR: AXIS 1**

| | | |
|---|---|---|
| J1 | 1 | AMPLIFIER SIGNAL / MOTOR DRIVE + |
| J1 | 2 | ENCODER PHASE A+ |
| J1 | 3 | LOGIC/SIGNAL GROUND |
| J1 | 4 | LIMIT SWITCH + |
| J1 | 5 | ENCODER PHASE A- |
| J1 | 6 | ENCODER INDEX + |
| J1 | 7 | COARSE HOME SWITCH |
| J1 | 8 | OPTICAL ISOLATOR RETURN |
| J1 | 9 | ENCODER POWER (+5 or +12 VDC) |
| J1 | 10 | ENCODER PHASE B+ |
| J1 | 11 | MOTOR DRIVE - |
| J1 | 12 | LIMIT SWITCH - |
| J1 | 13 | ENCODER PHASE B- |
| J1 | 14 | ENCODER INDEX - |
| J1 | 15 | AMPLIFIER ENABLE (ACTIVE HIGH) |

**J2: DC SERVO CONNECTOR: AXIS 2**

| | | |
|---|---|---|
| J2 | 1 | AMPLIFIER SIGNAL / MOTOR DRIVE + |
| J2 | 2 | ENCODER PHASE A+ |
| J2 | 3 | LOGIC/SIGNAL GROUND |
| J2 | 4 | LIMIT SWITCH + |
| J2 | 5 | ENCODER PHASE A- |
| J2 | 6 | ENCODER INDEX + |
| J2 | 7 | COARSE HOME SWITCH |
| J2 | 8 | OPTICAL ISOLATOR RETURN |
| J2 | 9 | ENCODER POWER (+5 or +12 VDC) |
| J2 | 10 | ENCODER PHASE B+ |
| J2 | 11 | MOTOR DRIVE - |
| J2 | 12 | LIMIT SWITCH - |
| J2 | 13 | ENCODER PHASE B- |
| J2 | 14 | ENCODER INDEX - |
| J2 | 15 | AMPLIFIER ENABLE (ACTIVE HIGH) |

Mating Connector: 15 pin D-subminiature male

J3: ANALOG AND DIGITAL I/O CONNECTOR

| | | |
|---|---|---|
| J3 | 1 | +5 VDC |
| J3 | 2 | ANALOG INPUT #4 |
| J3 | 3 | DIGITAL I/O, CHANNEL 16 |
| J3 | 4 | ANALOG INPUT #3 |
| J3 | 5 | DIGITAL I/O, CHANNEL 15 |
| J3 | 6 | " " , " 14 |
| J3 | 7 | " " , " 13 |
| J3 | 8 | " " , " 12 |
| J3 | 9 | " " , " 11 |
| J3 | 10 | " " , " 10 |
| J3 | 11 | " " , " 09 |
| J3 | 12 | " " , " 08 |
| J3 | 13 | " " , " 07 |
| J3 | 14 | " " , " 06 |
| J3 | 15 | " " , " 05 |
| J3 | 16 | " " , " 04 |
| J3 | 17 | " " , " 03 |
| J3 | 18 | " " , " 02 |
| J3 | 19 | " " , " 01 |
| J3 | 20 | ANALOG INPUT #2 |
| J3 | 21 | +12 VDC |
| J3 | 22 | ANALOG INPUT #1 |
| J3 | 23 | ANALOG REF. |
| J3 | 24 | GROUND |
| J3 | 25 | -12 VDC |
| J3 | 26 | GROUND |

Mating Connector: 26-pin dual-row IDC female, Circuit Assembly P/N CA-26IDS2-F-SPT or equivalent

**J4: RS-232 SERIAL INTERFACE CONNECTOR**

| | | |
|---|---|---|
| J4 | 1 | NC |
| J4 | 2 | DTR (OUTPUT) |
| J4 | 3 | TX (OUTPUT) |
| J4 | 4 | DSR (INPUT) |
| J4 | 5 | RX (INPUT) |
| J4 | 6 | RTS (OUTPUT) |
| J4 | 7 | CTS (INPUT) |
| J4 | 8 | 100 OHM PULL-UP TO +12 |
| J4 | 9 | GROUND |
| J4 | 10 | 100 OHM PULL-UP TO +12 |

Mating Connector: 10-pin dual-row IDC female, Circuit Assembly P/N CA-10IDS2-F-SPT or equivalent

**J5: EXTERNAL POWER CONNECTOR**

| | | |
|---|---|---|
| J5 | 1 | POWER: MOTOR 1 |
| J5 | 2 | POWER: MOTOR 2 |
| J5 | 3 | POWER: GROUND |
| J5 | 4 | POWER: + 5 VDC |

Mating Connector (assembly): Shell, AMP P/N 1-480424-0 Pins (4 each), AMP P/N 614731

**IBM-PC INTERFACE EDGE CONNECTOR**

| **Component Side** | **Solder Side** |
|---|---|
| A01 - NC | B01 - GROUND |
| A02 - BUS DATA 7 (HIGH TRUE) | B02 - RESET (ACTIVE HIGH) |
| A03 - BUS DATA 6 | B03 - NC |
| A04 - BUS DATA 5 | B04 - NC |
| A05 - BUS DATA 4 | B05 - NC |
| A06 - BUS DATA 3 | B06 - NC |
| A07 - BUS DATA 2 | B07 - -12 VDC |
| A08 - BUS DATA 1 | B08 - NC |
| A09 - BUS DATA 0 | B09 - +12 VDC |
| A10 - NC | B10 - NC |
| A11 - AEN | B11 - SMEMW (ACTIVE LOW) |
| A12 - BUS ADR. 19 (HIGH TRUE) | B12 - SMEMR (ACTIVE LOW) |
| A13 - BUS ADR. 18 | B13 - IOW (ACTIVE LOW) |
| A14 - BUS ADR. 17 | B14 - IOR (ACTIVE LOW) |
| A15 - BUS ADR. 16 | B15 - NC |
| A16 - BUS ADR. 15 | B16 - NC |
| A17 - BUS ADR. 14 | B17 - NC |
| A18 - BUS ADR. 13 | B18 - NC |
| A19 - BUS ADR. 12 | B19 - NC |
| A20 - BUS ADR. 11 | B20 - NC |
| A21 - BUS ADR. 10 | B21 - NC |
| A22 - BUS ADR. 09 | B22 - NC |
| A23 - BUS ADR. 08 | B23 - IRQ5 |
| A24 - BUS ADR. 07 | B24 - NC |
| A25 - BUS ADR. 06 | B25 - NC |
| A26 - BUS ADR. 05 | B26 - NC |
| A27 - BUS ADR. 04 | B27 - NC |
| A28 - BUS ADR. 03 | B28 - NC |
| A29 - BUS ADR. 02 | B29 - +5 VDC |
| A30 - BUS ADR. 01 | B30 - NC |
| A31 - BUS ADR. 00 | B31 - GROUND |

Note: When PC interface is not used, connect B11,B12,B13 and B14 to +5V through 4.7K ohm resistor. Also connect B2, A2 - A9 and A11 - A31 to ground.

**CONFIGURATION JUMPERS**

JP1 RESET SOURCE
Connect pins 1 and 2 for resistor/capacitor reset.
Connect pins 2 and 3 for IBM-PC bus reset. **(Factory setting)**

JP2 SIGNAL/MOTOR DRIVE SELECT, AXIS 1
Connect pins 1 and 2 for motor drive on pin 1 of J1. **(Factory setting)**
Connect pins 2 and 3 for analog signal on pin 1 of J1.

JP3 RAM CE SELECT
Connect pins 1 and 2 when battery used to backup RAM. **(Factory setting)**
Connect pins 2 and 3 when battery is not used.

JP4 SIGNAL/MOTOR DRIVE SELECT, AXIS 2
Connect pins 1 and 2 for motor drive on pin 1 of J2. **(Factory setting)**
Connect pins 2 and 3 for analog signal on pin 1 of J2.

JP5 Does not exist

JP6 EXTERNAL POWER/PC BUS +12 SELECT: AXIS 1
Connect pins 1 and 2 to use external power supply for PWM driver.
Connect pins 2 and 3 to use PC power supply for PWM driver. **(Factory setting)**

JP7 EXTERNAL POWER/PC BUS +12 SELECT: AXIS 2
Connect pins 1 and 2 to use external power supply for PWM driver.
Connect pins 2 and 3 to use PC power supply for PWM driver. **(Factory setting)**

JP8 COARSE HOME SELECT: AXIS 1
Connect pins 1 and 2 for TTL level input.
Connect pins 2 and 3 for optically isolated input. **(Factory setting)**

JP9 LIMIT+ SELECT: AXIS 1
Connect pins 1 and 2 for TTL level input.
Connect pins 2 and 3 for optically isolated input. **(Factory setting)**

JP10 LIMIT- SELECT: AXIS 1
Connect pins 1 and 2 for TTL level input.
Connect pins 2 and 3 for optically isolated input. **(Factory setting)**

JP11 COARSE HOME SELECT: AXIS 2
Connect pins 1 and 2 for TTL level input.
Connect pins 2 and 3 for optically isolated input. **(Factory setting)**

JP12 LIMIT+ SELECT: AXIS 2
Connect pins 1 and 2 for TTL level input.
Connect pins 2 and 3 for optically isolated input. **(Factory setting)**

JP13 LIMIT- SELECT: AXIS 2
Connect pins 1 and 2 for TTL level input.
Connect pins 2 and 3 for optically isolated input. **(Factory setting)**

JP14 ANALOG INPUT REFERENCE SELECT
Connect pins 1 and 2 if reference is supplied on pin 23 of J3. **(Factory setting)**
Connect pins 2 and 3 to use on board reference.

JP15 ENCODER PHASE A SINGLE ENDED: AXIS 1
Connect pins 1 and 2 for single ended encoder (signal in on pin 2 of J1). **(Factory setting)**
Disconnect pins 1 and 2 if encoder input is differential.

JP16 ENCODER PHASE B SINGLE ENDED: AXIS 1
Connect pins 1 and 2 for single ended encoder (signal in on pin 10 of J1). **(Factory setting)**
Disconnect pins 1 and 2 if encoder input is differential.

JP17 ENCODER INDEX ACTIVE LEVEL SELECT: AXIS 1
Connect pins 1 and 2 for single ended active high index (signal on pin 6 of J1). **(Factory setting)**
Connect pins 2 and 3 for single ended active low at index (signal on pin 14 of J1).
Disconnect pins 1,2 and 3 if index is differential, signal on pin 6 of J1 should go high and pin 14 of J1 should go low at index mark.

JP18 ENCODER INDEX ACTIVE LEVEL SELECT: AXIS 2
Connect pins 1 and 2 for single ended active low index (signal on pin 14 of J2).
Connect pins 2 and 3 for single ended active high index (signal on pin 6 of J2). **(Factory setting)**
Disconnect pins 1,2 and 3 if index is differential, signal on pin 6 of J2 should go high and pin 14 of J2 should go low at index mark.

JP19 ENCODER PHASE A SINGLE ENDED: AXIS 2
Connect pins 1 and 2 for single ended encoder (signal on pin 2 of J2). **(Factory setting)**
Disconnect pins 1 and 2 if encoder input is differential.

JP20 ENCODER PHASE B SINGLE ENDED: AXIS 2
Connect pins 1 and 2 for single ended encoder (signal on pin 10 of J2). **(Factory setting)**
Disconnect pins 1 and 2 if encoder input is differential.

JP21 ENCODER POWER SELECT: AXIS 1
Connect pins 1 and 2 for +5 VDC on pin 9 of J1.
Connect pins 2 and 3 for +12 VDC on pin 9 of J1. **(Factory setting)**

JP22 ENCODER POWER SELECT: AXIS 2
Connect pins 1 and 2 for +5 VDC on pin 9 of J2.
Connect pins 2 and 3 for +12 VDC on pin 9 of J2. **(Factory setting)**

JP23 - JP30 PC INTERFACE ADDRESS SELECTION, refer also to section 3.1.
Connect pins 1 and 2 for **1** on address line.
Connect pins 2 and 3 for **0** on address line.

| JUMPER | MEMORY ADDX | I/O ADDRX | FACTORY SETTING |
|---|---|---|---|
| JP23 | A19 | A9 | 1 to 2 |
| JP24 | A18 | A8 | 1 to 2 |
| JP25 | A17 | A7 | 2 to 3 |
| JP26 | A16 | A6 | 2 to 3 |
| JP27 | A15 | A5 | 2 to 3 |
| JP28 | A14 | A4 | 2 to 3 |
| JP29 | A13 | A3 | 2 to 3 |

| | | | |
|---|---|---|---|
| JP30 | A12 | A2 | 2 to 3 |

JP31 MEMORY / I/O MAP SELECTION: PC INTERFACE WRITE LINE
Connect pins 1 and 2 if DC2 is to appear in PC's memory map (74LS688 mounted in U14A position).
Connect pins 2 and 3 if DC2 is to appear in PC's I/O map (74LS688 mounted in U14B position). **(Factory setting)**

JP32 MEMORY / I/O MAP SELECTION: PC INTERFACE READ LINE
Connect pins 1 and 2 if DC2 is to appear in PC's memory map (74LS688 mounted in U14A position).
Connect pins 2 and 3 if DC2 is to appear in PC's I/O map (74LS688 mounted in U14B position). **(Factory setting)**

JP33 ENCODER PHASING: AXIS 1
JP34 Connect JP33 pins 1 and 2, JP34 pins 1 and 2 for standard phasing. **(Factory setting)**
Connect JP33 pin 1 to JP34 pin 1, and JP33 pin 2 to JP34 pin 2 for reverse phasing.

JP35 ENCODER PHASING: AXIS 2
JP36 Connect JP35 pins 1 and 2, JP36 pins 1 and 2 for standard phasing. **(Factory setting)**
Connect JP35 pin 1 to JP36 pin 1, and JP35 pin 2 to JP36 pin 2 for reverse phasing.

JP37 MICRO-CONTROLLER POWER SELECT
Connect pins 1 and 2 when battery is used to backup micro-controller (this option is not supported at this time).
Connect pins 2 and 3 when battery is not used. **(Factory setting)**

JP38 RAM POWER SELECT
Connect pins 1 and 2 when battery is used to backup RAM (connect JP3 pins 1 and 2).
Connect pins 2 and 3 when battery is not used (connect JP3 pins 2 and 3). **(Factory setting)**

JP39 SERIAL INTERFACE DTR / NETWORK SELECT
JP40 Connect JP39 pins 1 and 2, JP40 pins 1 and 2 for networked serial interface (TX output will go tri-state when board not selected, DTR output always true).
Connect JP39 pin 1 to JP40 pin 1, and JP39 pin 2 to JP40 pin 2 for non-networked serial interface (TX output always active, DTR output true when DC2 ready for data). **(Factory setting)**

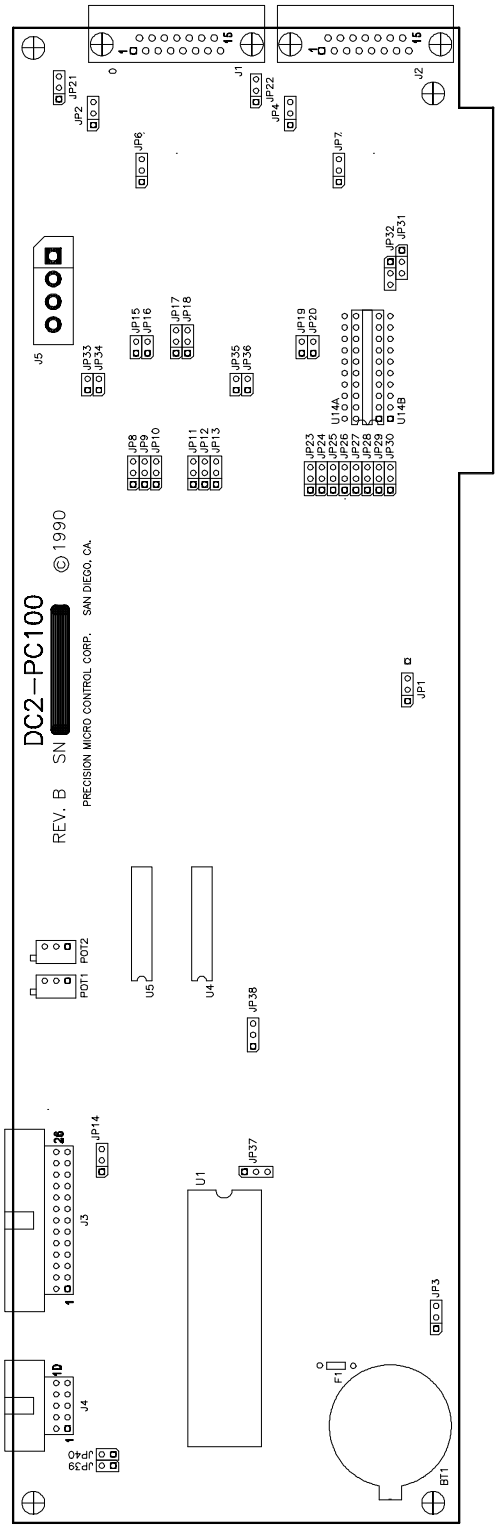


Figure 16: DC2 jumpers and connectors

# APPENDIX C: BINARY MODE COMMAND INTERFACE

In order to achieve the fastest PC to **DC2** command throughput, the **DC2** supports a "Binary Mode" command interface. This technique allows the PC to write binary coded commands directly into the **DC2**'s command buffer, thus bypassing the ASCII command interpreter. Using this method, any command replies the **DC2** generates will be also be coded in binary format.

The procedure for using this interface is as follows:

1. Place the **DC2** in binary mode by writing to the ATTENTION register.

2. Wait for the BUSY EXECUTING COMMAND flag in the STATUS register to be cleared by the **DC2**. Next wait for the READY FOR INPUT flag to be set. Then write in the number of bytes to be written to the command buffer (i.e., # of commands x 6) into the DATA IN register. Wait for the BUSY EXECUTING COMMAND flag in the STATUS register to be set by the **DC2**.

3. Write in from 1 to 40 commands into the DATA IN register using the following format for each command:

   Byte 1 command code
   Byte 2 axis # (binary 0-8, 0 for all, bit 7 set for register parameter)
   Byte 3-6 command parameter (binary, LSB first)

   The READY FOR INPUT flag should be set prior to writing each byte. When the last byte is written, the **DC2** will begin executing the commands.

3. If the READY WITH OUTPUT flag is set while the **DC2** is executing the command, the reply should be read from the reply buffer. The first byte read from the **DC2** will be the number of bytes in the reply buffer. Each command reply will have the following format:

   Byte 1 command code
   Byte 2 axis #
   Byte 3-6 reply value (binary, LSB first)

4. When the **DC2** clears the BUSY EXECUTING COMMAND status flag, it indicates that all commands have been executed. If it is necessary for the PC to terminate command execution before the **DC2** has set the flag, write to the ATTENTION register.

# APPENDIX D: SAMPLE PROGRAM IN 'C'

```
/*
 DC2 'C' Demo
 Precision Micro Control Corp. 1990
 Revision: 1.1a
 Date: 8/13/90

 Compiler: Microsoft Quick C version 1.0

 Abstract: This file contains a program for demonstrating the use  of the
DC2 high level language interface for the C programming language. The code
in this module must be compiled and then linked with the DC2 interface
code.  In order to build the program under Microsoft C, use the  following
command line:
       QCL /AM CDEMO.C DC2CIFC.OBJ
*/

include "dc2cifc.h"

#define ESC 0x1b

#define DC2ADDR 0x300           /* DC2 board base address. */

struct rpyfmt rpybuf[40];

main()
{
       char ky,axon;
       int axidx,axcnt;
       long xposn,yposn;

       /* Activate DC2 attention line to clear interface and
             place it in binary mode. */
       dc2attn(DC2ADDR);

       /* Put instructions on screen. */
       printf("DC2 interface library demo - hit ESCAPE key to exit.\r\n");
       printf("Cursor control keys cause axis 1 to jog by 1000 steps
if\r\n");
       printf("NUM LOCK is on, 1 step if it is off.  All other keys\r\n");
       printf("have no effect.\r\n");
       printf("Motor parameters should be set before this program is
run!\r\n\n");
                           /* Turn servos off, set positions to 0. */
       dc2cmd(DC2ADDR,2,MF,0L,DH,0L);
       axon = FALSE;

       while(1)
       {
             /* Get current servo positions from board. */
             dc2cmd(DC2ADDR,1,TP,0L);
             dc2rpy(DC2ADDR,sizeof(rpybuf), rpybuf);

             /* Display each position. */
             printf("AXIS 1: %11ld    AXIS 2: %11ld\r",
                                       rpybuf[0].val,rpybuf[1].val);


             if (kbhit())              /* Check for keypress. */
             {
                   if (!axon)  /* If motors are off, turn them on. */
                   {
                         dc2cmd(DC2ADDR,1,MN,0L);
```

```
                        axon = TRUE;
                    }
                    if((ky=getch()) == ESC)   /* Exit if ESCAPE hit. */
                    {
                        terminate();
                        exit(0);
                    }
                    else if(ky==0)      /* If keycode = 0, read again. */
                        ky=getch();
                        /* Set move distance in parameter array.*/
                    if (ky==0x4B)                    /* Left or down arrow.
*/
                        xposn = -1;
                    else if (ky==0x50)        /* Left or down arrow. */
                        yposn = -1;
                    else if (ky==0x4D)        /* Up or right arrow. */
                        xposn = 1;
                    else if (ky==0x48)        /* Up or right arrow. */
                        yposn = 1;
                    else if (ky=='4')         /* Left or down arrow. */
                        xposn = -1000;
                    else if (ky=='2')         /* Left or down arrow. */
                        yposn = -1000;
                    else if (ky=='6')         /* Up or right arrow. */
                        xposn = 1000;
                    else if (ky=='8')         /* Up or right arrow. */
                        yposn = 1000;
                    else
                        xposn = yposn = 0;

                    /* Execute relative move operation. */
                    if (xposn)
                    {
                        dc2cmd(DC2ADDR,1,AXIS1+MR,xposn);
                        xposn = 0;
                    }
                    else if (yposn)
                    {
                        dc2cmd(DC2ADDR,1,AXIS2+MR,yposn);
                        yposn = 0;
                    }
                }
            }
}


terminate()
{
        /* Put DC2 back in Decimal mode before exiting. */
        dc2cmd(DC2ADDR,4,DM,0L,EN,0L,XN,62L,XF,10L);
        /* Read the Xon character sent after DC2 finishes
              previous command. */
        while(dc2busy(DC2ADDR));
        if (dc2rdyout(DC2ADDR))
              putch(getdc2(DC2ADDR));
}
```

## APPENDIX E: SAMPLE PROGRAM IN BASIC

```
'DC2 demo program in Microsoft Quick Basic
' Version 1.0a
' 7/20/90
'Precision Micro Control Corp.
'
'
'Requires Quick library DC2BIFC.QLB loaded ' use: QB /LDC2BIFC.QLB BASDEMO
'
'Library built with:
' LINK /Q DC2BIFC.OBJ,DC2BIFC.QLB,,BQLB40.LIB;
' LIB DC2BIFC.LIB+DC2BIFC.OBJ;
'
'Revision History:
' 7/20/90  1.0a   Initial Release
'
'
'Declare procedures used to access DC2.
' Note: parameters passed by VALUE
'
'dc2attn(baseaddress) -
' Write to DC2 attention register. This will place DC2 command
' interface in binary mode and/or clear interface.
DECLARE SUB dc2attn (BYVAL dc2io%)
'
'dc2cmd(baseaddress,commandcnt,cmdbufoffset,cmdbufsegment) -
' Write commands in PC buffer to DC2. Assumes command interface
' is in binary mode.
DECLARE SUB dc2cmd (BYVAL dc2io%, BYVAL count%, BYVAL bufoff%,
      BYVAL bufseg%)
'
'dc2rpy(baseaddress,rpybufoffset,rpybufsegment) -
' Read reply from DC2 into PC buffer. Assumes command interface
' is in binary mode.
DECLARE FUNCTION dc2rpy% (BYVAL dc2io%, BYVAL bufoff%,
      BYVAL bufseg%)
'
'
CONST MAXCMDS = 40        'Maximum number of commands that can
                                ' be sent
CONST dc2base = &H300     'DC2 base I/O address
'
'DC2 command codes
CONST MF = 17
CONST MN = 19
CONST MR = 21
CONST TP = 82
CONST TT = 84

'Each DC2 command has the following format
TYPE CMDFMT
      axcmd AS INTEGER    'byte 1 command, byte 2 axis
      arg AS LONG         'bytes 3 through 6 long parameter
END TYPE
'
'Each DC2 reply has the following format
TYPE RPYFMT
      axcmd AS INTEGER    'byte 1 command, byte 2 axis
      value AS LONG             'bytes 3 through 6 Long value
END TYPE
'
'Create a buffer of consecutive DC2 commands
DIM cmdbuf(0 TO MAXCMDS) AS CMDFMT
```

```
'
'Create a buffer for DC2 replies
DIM rpybuf(0 TO 1) AS RPYFMT
'
PRINT "DC2 Basic Interface Demo Program    Rev. 1.0a"
PRINT "Precision Micro Control Corp."
'
'Call SUB to place DC2 in binary mode by accessing attention
' register
CALL dc2attn(dc2base)
'
MLOOP:
'
cmdbuf(0).axcmd = &H100 + MR    'Place Axis 1 Move in buffer
cmdbuf(0).arg = 1000
cmdbuf(1).axcmd = &H200 + MR    'Place Axis 2 Move in buffer
cmdbuf(1).arg = -1000
cmdbuf(2).axcmd = TT    'Place Tell Target command in buffer
cmdbuf(2).arg = 0
'
'Call SUB to issue commands in buffer to DC2
CALL dc2cmd(dc2base, 3, VARPTR(cmdbuf(0)), VARSEG(cmdbuf(0)))
'
'Call FUNCTION dc2rpy to read reply from DC2 into buffer
retnd% = dc2rpy(dc2base, VARPTR(rpybuf(0)), VARSEG(rpybuf(0)))
'
'Display axis positions from reply buffer
FOR axis% = 0 TO (retnd% \ 6) - 1
      PRINT "AXIS "; axis% + 1; " : "; rpybuf(axis%).value
NEXT
'
GOTO MLOOP
'
END
```

# APPENDIX F: DEFAULT SETTINGS & ERROR CODES

**DC2** Default Settings

| Parameter (Command) | Default |
|---|---|
| Mode (PM,VM,GM,QM,SM) | Position |
| Syncronization (SN,NS) | Off |
| Position (DH,FI,FE) | 0 |
| Velocity (SV) | 1000000 |
| Acceleration (SA) | 5000 |
| Deceleration (DS) | 5000 |
| Proportional Gain (SG) | 200 |
| Derivative Gain (SD) | 200 |
| Integral Gain (SI) | 5 |
| Integral Limit (IL) | 50 |
| Derivative Sampling Period (FR) | 0 |
| Velocity Gain (VG) | 0 |
| Vector Velocity (VV) | 1000000 |
| Vector Acceleration (VA) | 5000 |
| Vector Deceleration (VD) | 5000 |
| Maximum Torque (SQ) | 127 (setting for DC2-PC110) |
| Maximum Following Error (SE) | 1024 |
| Output Phasing (PH) | 0 |
| Limit Switch Mode (LM) | 0 |
| Limit Switches (LN,LF) | Off |

**DC2** Error Codes

| | |
|---|---|
| Parameter Error | -1 |
| Command Not Valid | -2 |
| Negative Repeat Value | -3 |
| Macro Define Command Error | -4 |
| Macro Number Out of Range | -5 |
| Macro Does Not Exist | -6 |
| Command Canceled | -7 |

| | |
|---|---|
| Axis Number Out of Range | -8 |

# APPENDIX G: TROUBLESHOOTING GUIDE

Communications (PC compatible based DC2-PC)

| Symptom | Possible Cause | Solutions |
|---|---|---|
| DC2-PC board does not communicate with PMC Win Control or DC2CNTRL. | DC2-PC board is not being properly reset. | Verify that JP1 (reset source) is set to pins 2 and 3 (IBM-PC bus reset). |
| | DC2-PC board not properly addressed. | The communication software defaults to I/O address 300h. Refer to section 3.1 and Windows utilities 'MCSETUP.EXE' or DOS utilities 'INIT.DC2'. |
| | Address conflict between DC2-PC and another board installed in PC. | Identify the addresses of all boards installed in the PC. |
| | PC interrupt conflict (IRQ5) between DC2-PC and another PC device. | Change IRQ level for other device or lift pi 15 of U17 (DC2PAL R.A). |
| | Faulty PC motherboard edge connector. | Install the DC2-PC into a different PC motherboard edge connector |
| | Faulty PC | Install the DC2-PC into a different PC. |
| | Failed DC2-PC | Contact Precision MicroControl and ask for application assistance. |
| The DC2-PC board stops communicating with the PC. | Electrical noise caused the DC2-PC's micro controller to stop executing code. | Identify and eliminate the source of the noise. |

Communications (RS-232)

| Symptom | Possible Cause | Solutions |
|---|---|---|
| DC2-PC board does not communicate with Host. | DC2-PC board is not being properly reset. | Verify that JP1 (reset source) is set to pins 1 and 2 (power good reset). |
| | RS-232 interface not wired correctly. | Refer to section 2.6. |
| | Edge connector not terminated for stand alone operation. | Refer to appendix B (IBM interface edge connector). |
| | Stand alone power supply | Verify +5, +12, and -12 supply voltages on the terminating edge connector (refer to appendix B). |
| | Serial communications program not setup properly. | Refer to the communication software user's manual. If the software configuration appears correct install the DC2-PC in a PC compatible computer to verify the board is operational (using PMC WinControl or DC2CNTRL.EXE). |
| | Failed DC2-PC | Contact Precision MicroControl and ask for application assistance. |
| The DC2-PC board stops communicating with the host. | Electrical noise caused the DC2-PC's micro controller to stop executing code. | Identify and eliminate the source of the noise. |

## Position Reporting

| Symptom | Possible Cause | Solutions |
|---|---|---|
| Tell Position (aTP) command does not indicate changing position when motor shaft / encoder is rotated | Incorrect encoder power supply voltage | If encoder supply voltage is supplied by DC2-PC verify that JP21 and/or JP22 is set correctly (refer to appendix B). |
| | Incorrect encoder wiring | Refer the to wiring diagram in section 2.2. |
| | DC2-PC is configured for a differential encoder when a single ended encoder is being used. | Verify the settings for JP15 and/or JP16 (refer to appendix B). |

## Motion

| Symptom | Possible Cause | Solutions |
|---|---|---|
| When an axis is configured fpr analog control signal output, a motor moves prior to issuing the motor on (aMN) command. | An offset is present in the analog signal output. | Adjust the analog output potentiometer. |
| The motor does not try to 'hold' position after issuing the Motor oN (aMN) command. No motion is observed after issuing a move command. | The motor output jumper is not correctly configured. | Verify configuration of JP2 and/or JP4. Refer to appendix B. |
| | The servo is not 'tuned'. | Either the proportional gain is set too low or the derivative gain is set too high. Refer to sections 1.3 and 5.0. |
| | Check the status of the servo with the Tell Status (aTS) command. The motor on flag (bit 0) should be set to a '1'. | An error probably exists. This would be indicated by bits 1, 26, or 29. |
| Issuing the Motor oN (aMN) command causes the servo to oscillate or 'hum'. | The servo is not properly 'tuned'. | Either the proportional gain is set too high or the derivative gain is set too low. Refer to sections 1.3 and 5.0. |
| Issuing the Motor oN (aMN) command causes the servo to 'take off running'. | The motor is reversed phassed. | Refer to sections 2.2 and 5.0. |
| After issuing a move command motion is observed but the motor stops before the target position is reached. | The servo is not properly 'tuned'. A following error is indicated by bit 1 (motor error) in the servo status. | Either the proportional gain is set too low or the derivative gain is set too high. Refer to sections 1.3 and 5.0. |
| | A limit switch was sensed. Bit 26 or 29 of the servostatus is set to a '1'. | Issue a Motor oN command to clear the limit condition, issue a move command in the oposite direction to 'move off the limit'. |
| | The desired velocity (set by the Set Velocity command) is higher than the electrical/mechanical system components can achieve. Bit 1 (motor error) of the servo status is set to a '1'. | Determine the maximum velocity of the servo system. Reduce the desired velocity using the Set Velocity command. Refer to sections 1.3 and 5.0. |
| | The frequency of the encoder inputs is higher than the maximum encoder count rate (1 MHz). Bit 1 (motor error) of the servo status is set to a '1'. | Reduce the desired velocity using the Set Velocity command. Refer to sections 1.3 and 5.0. |
| | The Set torQue (aSQn) command is limiting the output. Bit 1 (motor error) of the servo status is set to a '1'. | Increase the value set by the Set torQue command. Refer to section 9.1. |
| After a move is commanded and completed (the target possition has been reached) the motor oscillates or 'hums'. | The servo is not properly 'tuned'. | Either the proportional gain is set too high or the derivative gain is set too low. Refer to sections 1.3 and 5.0. |

| | | |
|---|---|---|
| When an axis is configured fpr analog control signal output, a motor moves prior to issuing the motor on (aMN) command. | An offset is present in the analog signal output. | Adjust the analog output potentiometer. |
| The motor does not try to 'hold' position after issuing the Motor oN (aMN) command. No motion is observed after issuing a move command. | The motor output jumper is not correctly configured. | Verify configuration of JP2 and/or JP4. Refer to appendix B. |
| After a move is commanded and completed the current position does not equal the target possition. | The servo is not properly 'tuned'. | Increase the integration limit and the integral gain. Refer to sections 1.3 and 5.0. |

## Limits and Homing

| Symptom | Possible Cause | Solutions |
|---|---|---|
| Limit, Home, and/or Index inputs are not properly handled by the DC2-PC | Incorrect wiring. | Refer to section 2.2 , 2.3. 7.6, 7.7 and appendix B. |
| | Incorrect jumper configuration. | Verify proper jumpering for JP8 - JP13, JP17, JP18. Refer to appendix B. |
| | | Manually operate the suspect input and verify that the proper state is indicated by the servo status (aTS). |
| During commanded move the motor stops prematurally. | A limit condition is indicated by the servo status. No limit switch was tripped. | Electrical noise caused the DC2-PC to sense a 'phantom limit trip'. |
| Unexpected results are observed during a homing sequence. | Homing an axis with the Define Home command is not supported while an axis is in motion | To define a new home position while an axis is in motion use the Find Edge and/or Find Index commands. Refer to section 7.6. |
| During a 'homing' procedure the home position is incorrectly defined. | The DC2-PC sensed a home or index input at the wrong position. | Electrical noise caused the DC2-PC to sense a 'phantom home or index input'. |

## I/O

| Symptom | Possible Cause | Solutions |
|---|---|---|
| General purpose digital I/O not operating as expected. | Channel not properly configured as input, output, high true, or low true. | Refer to sections 8.0 and 9.5. |
| | Incorrect wiring. | Refer to section 2.4 and appendix B. |
| A channel configured as an output is not reaching valid TTL levels. | The current requirement of external device exceeds DC2-PC I/O spec's. | Refer to sections 1.1 (DC2-PC specs's) and 2.4 (Relay rack interface) |
| Analog input channel/channles not reporting expected values. | Incorrect wiring. | Refer to sections 2.5 and appendix B. |
| | The analog reference input is not properly configured. | Verify that JP14 is set to match the application (external or DC2-PC supplied analog reference). Refer to sections 2.5 and appendix B. |

General

| Symptom | Possible Cause | Solutions |
|---|---|---|
| The DC2-PC fails to responding within expected time frame after a command, command string, or macro is executed. | A conditional command (Wait for channel oN, Wait for Stop ...etc) did not recognize the specified event allowing the command to complete execution. | Enter ESCAPE (the key or ASCII code) to halt current command/string/macro. |
| Unexpected motor motion. | Commanded motor motion continues even after the escape key is entered. | The escape key only halts additional command execution. Once motion has begun the command has completed execution. The motion can be stopped with the STop or ABort commands. |
| The Tell Velocity command reports a value that does not equate to the current velocity. The application requires the capability to report the current velocity of an axis. | The Tell Velocity command reports the current desired velocity as calculated by the trajectory generator. | There is no command to report current 'actual velocity'. |
| The Version (VE) command indicates that the board is a DC2-PC140. | All DC2-PC's are originally configured as DC2-PC140's. | ---------- |
| A command/command string is not executed and the board returns a dash followed a number. | A command error was detected. | Refer to appendix F. |
| After cycling power, previously defined macros are no lomger resident in DC2-PC memory. | Battery backup circuit not operating properly. | Verify that JP3 and JP38 are configured for battery backup. |
| | | Verify that the battery voltage is within specs (+2.2 to +3.2 VDC). |
| | | Verify that fuse F1 has not blown. |
| The DC2-PC stops communicating, previously defined macros are no longer resident in DC2-PC memory. | Electrical noise corrupted the DC2-PC's memory. | Identify and eliminate the source of the noise. |

# INDEX