

NextMove PC

Installation Manual

Issue 1.0

Draft



© 1998. All rights reserved.

This manual is copyrighted and all rights are reserved. This document may not, in whole or in part, be copied or reproduced in any form without the prior written consent of Baldor Optimised Control.

Baldor Optimised Control makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of fitness for any particular purpose. The information in this document is subject to change without notice. Baldor Optimised Control assumes no responsibility for any errors that may appear in this document.

MINT™ is a registered trademark of Baldor Optimised Control Ltd.

Limited Warranty

For a period of one (1) year from the date of original purchase, BALDOR will repair or replace without charge controls which our examination proves to be defective in material or workmanship. This warranty is valid if the unit has not been tampered with by unauthorized persons, misused, abused, or improperly installed and has been used in accordance with the instructions and/or ratings supplied. This warranty is in lieu of any other warranty or guarantee expressed or implied. BALDOR shall not be held responsible for any expense (including installation and removal), inconvenience, or consequential damage, including injury to any person or property caused by items of our manufacture or sale. (Some states do not allow exclusion or limitation of incidental or consequential damages, so the above exclusion may not apply.) In any event, BALDOR's total liability, under all circumstances, shall not exceed the full purchase price of the control. Claims for purchase price refunds, repairs, or replacements must be referred to BALDOR with all pertinent data as to the defect, the date purchased, the task performed by the control, and the problem encountered. No liability is assumed for expendable items such as fuses.

Goods may be returned only with written notification including a BALDOR Return Authorization Number and any return shipments must be prepaid.

Baldor Optimised Control Ltd.

178-180 Hotwell Road

Bristol

BS8 4RP

U.K.

Tel: (+44) (117) 987 3100

FAX: (+44) (117) 987 3101

Safety Notice:

Only qualified personnel should attempt the start-up procedure or troubleshoot this equipment.

This equipment may be connected to other machines that have rotating parts or parts that are controlled by this equipment. Improper use can cause serious or fatal injury. Only qualified personnel should attempt to start-up, program or troubleshoot this equipment.

Operating Environment safety is discussed in Section 8.


Precautions:





WARNING: Do not touch any circuit board, power device or electrical connection before you first ensure that no high voltage present at this equipment or other equipment to which it is connected. Electrical shock can cause serious or fatal injury. Only qualified personnel should attempt to start-up, program or troubleshoot this equipment.





WARNING: Be sure that you are completely familiar with the safe operation of this equipment. This equipment may be connected to other machines that have rotating parts or parts that are controlled by this equipment. Improper use can cause serious or fatal injury. Only qualified personnel should attempt to program, start-up or troubleshoot this equipment.


 **WARNING:** Be sure that you are completely familiar with the safe programming of this equipment. This equipment may be connected to other machines that have rotating parts or parts that are controlled by this equipment. Improper programming of this equipment can cause serious or fatal injury. Only qualified personnel should attempt to program, start-up or troubleshoot this equipment.


 **WARNING:** Be sure all wiring complies with the National Electrical Code and all regional and local codes. Improper wiring may result in unsafe conditions.


 **WARNING:** The stop input to this equipment should not be used as the single means of achieving a safety critical stop. Drive disable, motor disconnect, motor brake and other means should be used as appropriate. Only qualified personnel should attempt to program, start-up or troubleshoot this equipment.

 **WARNING:** Improper operation or programming of the control may cause violent motion of the motor shaft and driven equipment. Be certain that unexpected motor shaft movement will not cause injury to personnel or damage to equipment. Peak torque of several times the rated motor torque can occur during control failure.

 **WARNING:** The motor shaft will rotate during the homing procedure. Be certain that unexpected motor shaft movement will not cause injury to personnel or damage to equipment.

 **CAUTION:** To prevent equipment damage, be certain that the input power has correctly sized protective devices installed.

 **CAUTION:** To prevent equipment damage, be certain that input and output signals are powered and referenced correctly.

 **CAUTION:** To ensure reliable performance of this equipment be certain that all signals to/from the controller are shielded correctly.

Manual Revision History

Issue	Date	BOCL Reference	Comments
1.0	May 1998	MN277-000	First release of Installation Manual bringing together: <ul style="list-style-type: none">• <i>NextMove PC</i> Overview• Getting Started Guide• <i>WorkBench</i> User Guide• <i>NextMove PC</i> Hardware Guide• Getting Started with CAN Peripherals

1.	Read Me First	1
2.	Key to Symbols Used in this Manual	3
3.	Hardware Features	5
3.1	Technical Specification	6
3.1.1	Machine Control I/O	7
3.1.2	Miscellaneous and Mechanical Specification.....	8
4.	Introduction to the <i>NextMove</i> Software	11
4.1	MINT Programming Language	11
4.2	MINT Programmers Toolkit.....	16
4.3	NextMove WorkBench	17
4.4	Keyword Summary	20
5.	Options	26
5.1	CAN Operator Panel (<i>KeypadNode</i>).....	27
5.2	CAN I/O Nodes.....	28
5.3	The Encoder Splitter/Buffer Board.....	29
5.4	Expansion Boards.....	29
5.5	Breakout Board	29
5.6	MINT Interface Library	30
6.	Getting Started	32
6.1	Installing the <i>NextMove PC</i> Controller.....	33
6.1.1	Before Inserting the Card.....	35
6.1.2	Placing the Card In the Computer.....	36

6.2	Installing The Software	37
6.2.1	Starting The WorkBench.....	37
6.2.2	Selecting A <i>NextMove PC</i>	38
6.2.3	Downloading an Application.	40
6.3	Minimum System Wiring Example	41
6.4	Setting Up A Servo Drive	44
6.4.1	Ground Connections.....	44
6.4.2	Encoder Connections.....	45
6.4.3	Amplifier Demand/Command Signals.....	46
6.4.4	Amplifier Enable	46
6.4.5	Limit And Stop Switches.....	47
6.4.6	Testing System Wiring	47
6.4.7	Checking The Encoder.....	47
6.4.8	Checking the Drive Enable	49
6.4.9	Checking Motor Polarity.....	49
6.4.10	Setting System Gains.....	51
6.4.11	Selecting Servo Loop Gains	55
6.4.12	System Gains for Current Control by Empirical Method.....	56
6.4.13	Mathematical Method Of Calculating System Gains.....	57
6.4.14	System Gains for Velocity Control	57
6.4.15	Fine Tuning System Gains	59
6.4.16	Eliminating Steady-State Errors.....	61
6.5	Setting Up A Stepper Drive.....	62
6.5.1	Limit And Stop Switches.....	62
6.5.2	Pulse Generator Outputs.....	62
6.5.3	Amplifier Enable	62

6.5.4	Ground Connections	62
6.5.5	Testing System Wiring	63
6.6	Input/Output.....	65
6.6.1	Changing the I/O Configuration	66
6.6.2	Example I/O Configuration.....	67
7.	Introduction to the MINT™ Programming Language	72
7.1	The Configuration File	72
7.2	The First MINT Program	74
7.2.1	Program Narrative	76
7.3	A Simple Cut to Length Feeder.....	77
7.3.1	<i>Configuration Buffer</i> FEEDER.CFG	78
7.3.2	<i>Program buffer</i> FEEDER.MNT	80
7.3.3	Cut To Length Program Narrative	84
7.3.4	Using Batch Numbers	87
7.4	X-Y Teach and Replay Program.....	87
7.5	Software Gearbox Example - Coil Winding Machine.....	90
7.5.1	Program Narrative	91
7.5.2	Remote Operation Using the COMMS array.....	91
7.6	Infeed Packaging Machine	92
8.	Hardware Guide	96
8.1	Operating Environment	96
8.2	Connection to the PCB.....	98
8.3	Digital I/O	99
8.3.1	Digital Inputs.....	99
8.3.2	Fast Interrupt	101

8.3.3	Digital Outputs.....	102
8.4	Analog I/O	104
8.4.1	Analog Inputs	104
8.4.2	Analog Outputs (Drive Demand/Command).....	106
8.5	Encoder Interface	106
8.6	Error Relay	107
8.7	Stepper Drive Outputs	108
8.8	Communications.....	109
8.8.1	Serial Port.....	109
8.8.2	CAN Bus.....	110
8.9	Reset State.....	112
8.9.1	Communications.....	112
8.9.2	Digital Outputs.....	112
8.9.3	Analog Outputs.....	113
8.9.4	Pulse Generator Outputs.....	113
8.10	LEDs.....	113
8.11	Expansion Board.....	114
8.12	Miscellaneous	116
8.12.1	Emulator Connection	116
8.12.2	System Watchdog.....	116
8.12.3	Interrupt Level.....	116
9.	CE Marking.....	118
9.1	Directives.....	118
9.1.1	Machinery Directive 89/392/EEC	118
9.1.2	Low Voltage Directive 72/23/EEC.....	118
9.1.3	EMC Directive 89/336/EEC	118

9.2	EMC Performance of <i>NextMove PC</i>	119
9.3	Conditions of CE Marking	119
10.	Getting Started with CAN Peripherals	121
10.1	Network Possibilities	122
10.2	Quick Start	122
10.2.1	Jumper settings	123
10.2.2	Connections and Configuration.....	124
10.3	NextMove PC and CAN Peripherals.....	125
10.3.1	Selection of CAN Channel	125
10.3.2	Selection of CAN Baud Rate	126
10.3.3	Selection Of Node ID	126
10.3.4	Network Termination	128
10.3.5	Static Configuration	129
10.3.6	Normal Operation	132
10.4	An Example Network.....	132
10.5	Using A <i>KeypadNode</i>	134
10.6	MINT Keyword Summary	135
11.	MINT Programmer's Toolkit.....	137
11.1	Starting The WorkBench	137
11.2	The Main Toolbar	138
11.3	Selecting the Board Address.....	139
11.4	Downloading an Application.....	141
11.5	Terminal	141
11.6	Program/Configuration Editors.....	143
11.7	Squash	144

11.8	Macros	145
11.9	Setting up Gains.....	147
11.10	Motion WatchWindow.....	147
11.11	The Comms Window.....	148
11.12	Dual Port Ram WatchWindow.....	149
11.13	Software Oscilloscope.....	150
11.14	Axis WatchWindow.....	152
11.15	Digital I/O.....	153
11.16	Analog Inputs.....	154
11.17	Project Files.....	155
11.18	DOS Utilities.....	156
12.	Trouble Shooting Guide.....	158
13.	Breakout Board.....	162
13.1	Encoders: P1, P2, P3 and P4.....	164
13.2	CAN: JP7 and JP8.....	164
13.3	Terminal Block JP5.....	165
13.4	Terminal Block JP6.....	166
14.	Bibliography.....	170

1. Read Me First




This manual contains information for installing and commissioning the *NextMove PC* 8 axis intelligent servo motor controller. The manual is split into the following main sections:

- **Overview:** Sections 3 through to 5. This contains an overview of the *NextMove PC* controller, MINT™, the motion programming language and options available for *NextMove PC*.
- **Getting Started:** Sections 6. This section contains details of getting started with *NextMove PC*, checking wiring and tuning the motors.
- **Introduction to MINT Programming Language:** Section 7. This provides an overview of the MINT motion programming language.
- **Hardware Guide:** Sections 8 through to 9 provide a detailed description of the hardware interfaces on *NextMove PC*. For example, the digital and analog I/O is discussed.
- **Getting Started with CAN Peripherals:** Section 10. This provides a description of configuring *NextMove PC* with a number of CAN Peripherals.
- **WorkBench:** Section 10. Provides information on *WorkBench* for *NextMove PC*.

It is advisable to read Section 6, Getting Started, if unfamiliar with *NextMove PC*.

2. Key to Symbols Used in this Manual

Throughout this section various icons and conventions are used to indicate specific functions:

-  The screwdriver icon indicates that it is necessary to make a physical connection to *NextMove PC* by way of screw terminations.
-  The disk icon together with filename is used to indicate that a MINT program (the motion control language used to program *NextMove PC*) should be downloaded to the controller. The filename indicates the name of the *buffer*.
-  The prompt icon indicates that the following commands should be typed in directly to the terminal at the MINT **P>** or **C>** prompt.

[Ctrl]+[E] Type **Ctrl** and **E** at the same time.

3. Hardware Features

NextMove PC is an 8 axis intelligent servo motor controller for use in ISA bus based PC systems. The card has 1k words of Dual Port RAM (DPR) for rapid and easy transfer of data to and from the host.

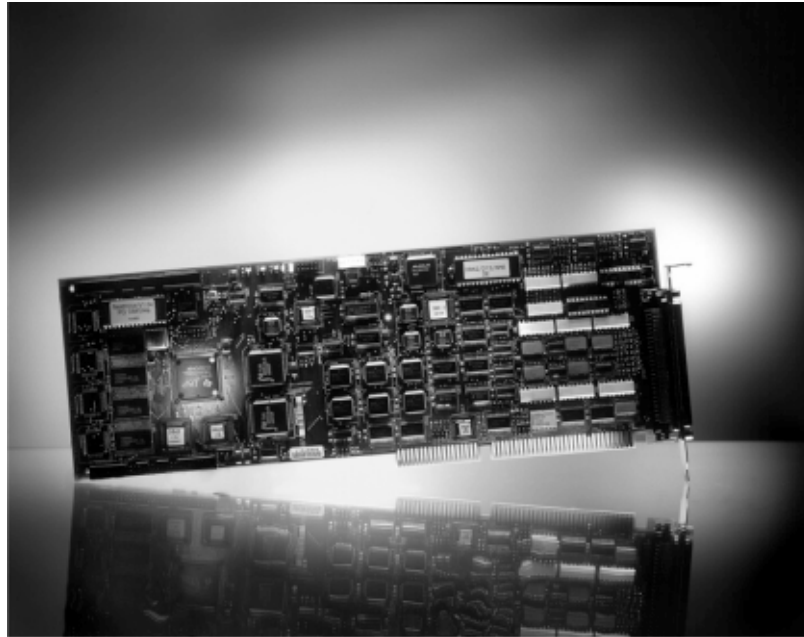


Figure 1: *NextMove PC* Controller

Features:

- Floating point Digital Signal Processor at the core of the *NextMove PC* controller which is capable of executing 33 million floating point instructions per second.
- Four axes of servo control (axes 0-3) with 12 bit analog outputs and incremental encoder feedback; 500 or 1000 microsecond servo loop update rate.
- Four axes of stepper motor control (axes 4-7) which can be addressed concurrently with the servos, open collector step and direction outputs to stepper motor drives.
- 16 bit ISA bus interface via 2Kbyte dual ported RAM
- 512 Kbytes Volatile RAM.
- Incremental encoder feed back, with ability to latch encoder position within 30 μ s.

- 24 inputs, 12 outputs opto-isolated PNP or NPN (jumper configured). Software configurable as limit inputs, home inputs, enable outputs for amplifiers or general purpose I/O.
- Eight 12 bit analog inputs, jumper configurable as 4 differential inputs. Readable as 0-5V, $\pm 2.5V$ or $\pm 10V$.
- Control Area Network (CAN) channels at 1MBit per second for connection of option boards, drive modules and operator panels up to 40m away.
- 32 bit bus expansion via two 50-way headers. Used for axis expansion.
- Voltage free contact for error output.
- Connections brought out via high density 100 way connector suitable for high density ribbon cable, round-flat cable or individual wires.
- DIN-rail mount break-out board with screw terminations for I/O and 'D' shells for encoders.

3.1 Technical Specification

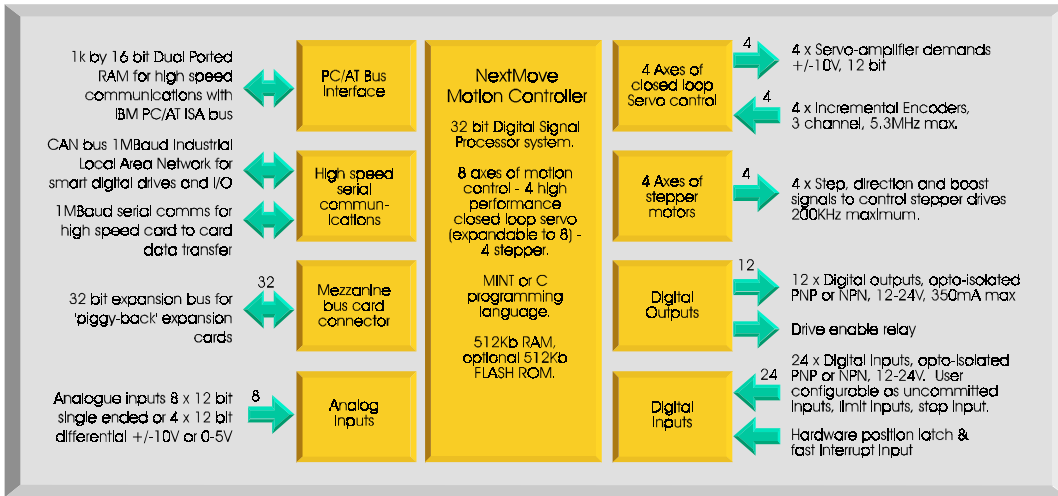


Figure 2: Technical Overview

3.1.1 Machine Control I/O

<p>User Digital Inputs:</p>	<ul style="list-style-type: none"> • 24 input lines. • PNP or NPN opto-isolated (factory set). • 12 to 24V $\pm 20\%$ on, 3V max off. • Software programmable as limit, home, stop or error inputs for any axis.
<p>User Digital Outputs:</p>	<ul style="list-style-type: none"> • 12 output lines driven by Darlington array. • 1 bank of 8 (0-7) and 1 bank of 4 (8-11). • PNP or NPN (optically) opto-isolated (factory set per bank). • Software programmable as drive enable or error output for any axis. • 50mA continuous source on all channels. 750mA total per bank
<p>Fast Interrupt:</p>	<ul style="list-style-type: none"> • 1 input per controller. • 5V CMOS, capacitively decoupled. • Latches position of all axes within 30 microseconds on falling edge.
<p>Enable (Relay) Output:</p>	<ul style="list-style-type: none"> • Voltage free change over relay rated at 150mA @ 24V dc. • Contact sensitivity 10μA @ 10mV DC • 1 output per controller, software programmable. • Fail safe operation: relay de-energised on an error or loss of power.
<p>Analog Inputs:</p>	<ul style="list-style-type: none"> • 8 independent analog channels. • 12 bit resolution. • Jumper selectable for voltage range (0-5/± 2.5V and ± 10V). • Software selectable for differential or single ended operation.
<p>Analog Outputs:</p>	<ul style="list-style-type: none"> • 1 output per axis for motor demand/command signal (software configurable). • ± 10V output ($\pm 0.1\%$). • 12 bit resolution (4.88mV per bit).

Encoder Inputs:	<ul style="list-style-type: none"> • Encoder input per axis for positional feedback via incremental encoder. • Operates with both single ended (TTL or open collector) or differential (TTL or RS422) output type. Differential is recommended. • Accepts three channel incremental encoders (A, B and Z). A and B channels are quadrature decoded. • Minimum requirement: Channel A and B single ended TTL. • Maximum encoder frequency: 5.3MHz quadrature count.
Stepper outputs:	<ul style="list-style-type: none"> • Step, direction and boost. • Open collector outputs. 100mA @ 24V. • 200kHz maximum output frequency.

3.1.2 Miscellaneous and Mechanical Specification

Power Input:	<ul style="list-style-type: none"> • +5V at 850mA • ±12V at 250mA • User Power: +12V to 24V at 850mA
Weight:	<ul style="list-style-type: none"> • Approximately 305g / 0.67lb
Operating Temperature:	<ul style="list-style-type: none"> • 0 - 40°C / 32 - 104°F Ambient
Warranty:	<ul style="list-style-type: none"> • 5 year return to manufacturer.

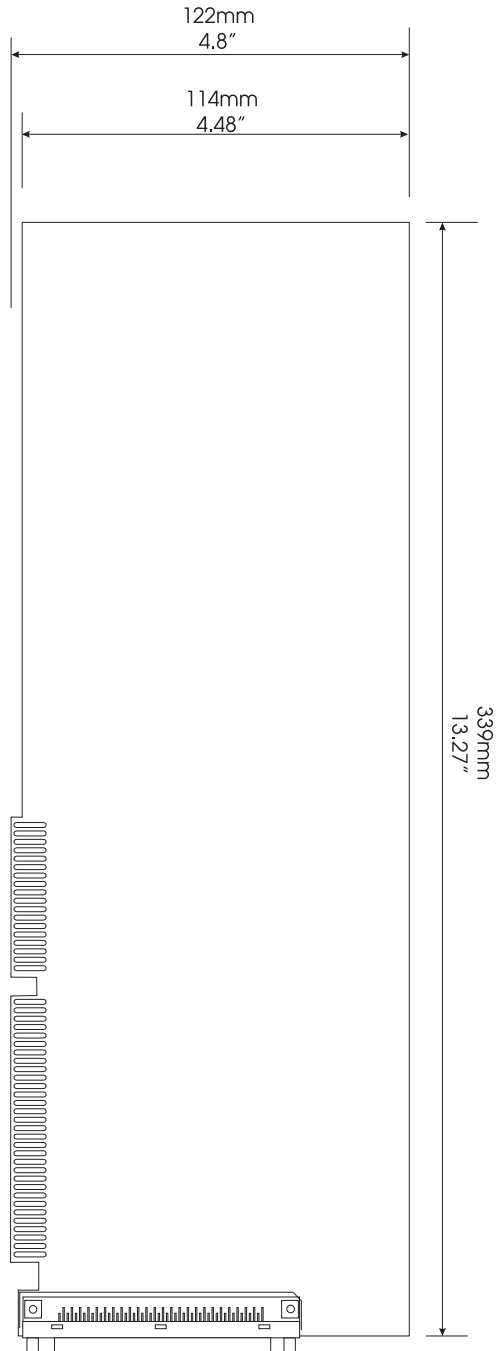


Figure 3: *NextMove PC* Overall Dimensions

4. Introduction to the *NextMove* Software

There are two main programs supplied with *NextMove PC*. These are the *NextMove PC WorkBench* and the motion control language, MINT.

4.1 MINT Programming Language

MINT is a structured form of *Basic* which has been custom designed for motion control applications. The MINT language has been written to allow users to get started quickly and run simple motion programs, also providing a wide range of more powerful commands for complex applications. MINT's onboard line editor and command line interface allows simple programs to be created quickly with only the aid of a terminal emulator. *NextMove WorkBench* is a pre-configured Windows terminal emulator supplied with *NextMove PC*. The installation CD can be found attached with this manual.

MINT is used within thousands of applications worldwide, servicing many high demand industries such as textiles and packaging. Applications range from simple single axis applications to complex multi-axis, multi-controller applications via a host-controlled link. It is MINT's flexible and powerful command set that is able to provide a solution to the vast number of industrial motion control applications. Some examples of these include:

- Packaging machines
- Multi-axis screen printers
- Print registration
- High precision test machines
- Spin welding
- Textile robots
- CNC machines

MINT Features for *NextMove PC*:

- Support for 8 axis of control on one card
- Support for both servo and stepper motors
- Basic constructs such as **IF . . THEN**, **FOR . . NEXT** and subroutines.
- User variables as with Basic. Variables can be given any meaningful name up to 10 characters in length
- Subroutines that are referenced with a label rather a line number
- Interrupts on inputs
- Array variables. Size limited only by available memory
- Error recovery from *limits, following errors* or external errors

- Circular and linear interpolation
- Flying Shears
- Cam profiling and Spline moves
- Position or velocity locked encoder following
- Tokenised (semi-compiled) source code at run time to give faster program execution speed
- Ability to define Macros for speed and program readability
- Floating point maths with comprehensive maths library
- Full software control of all the I/O and safety features of *NextMove PC*.

In addition to usual *Basic* type commands such as **PRINT**, **FOR . .NEXT** and **IF . .THEN**, MINT has a number of keywords dedicated to Motion Control and input/output. For example, keywords are provided for:

- Speed and positional control.
- Torque control
- Linear and Circular interpolation.
- Cam profiling and flying shears.
- Spline moves

There is also full control over basic motor control parameters such as servo loop gains in addition to all the digital and analog I/O on the controller.

MINT splits motion keywords into 2 categories, motion variables and motion commands. Motion variables are keywords that set axis parameters. These can be read or written (although not necessarily both). For example:

```
SPEED[0,1] = 10,20
ACCEL[0,1] = 100,200
MOVER[0,1] = 50,100 : GO[0,1]
```

The use of the square brackets controls 2 axes in the system, where axis 0 is the first axis followed by axis 1 and so on. The use of the square brackets is optional as MINT is very flexible in its multi-axis syntax. In this example; axis 0 is set up with a speed of 10 units/s, acceleration of 100 units/s² and a relative move of 50 units and axis 1 is set up with a speed of 20 units/s, acceleration of 200 units/s² and a relative move of 100 units. **SPEED**, **ACCEL** and **MOVER** are all motion variables. The motion command **GO** is used to initiate motion. Most motion keywords, they can be abbreviated to 2 or 3 letters, for example: **SPEED** can be replaced with **SP** and **MOVER** replaced with **MR**. This saves both on code space and typing. The example above could be abbreviated to:

```
SP[0,1]=10,20
AC[0,1]=100,200
MR[0,1]=50,100:GO[0,1]
```

The utility called *Squash* within *NextMove WorkBench* enables programs to be compressed by replacing keywords with their abbreviated counterparts. Refer to Section 11.7.

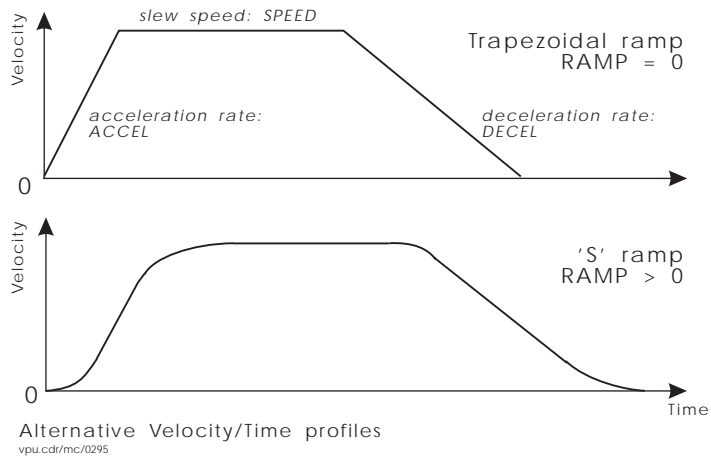


Figure 4: Trapezoidal move with S ramping applied

Moves are based on a trapezoid, which have separately configurable acceleration and deceleration parameters. MINT also supports 'S' curve profiles by use of the RAMP keyword.

At any time during motion, the position of the motor can be printed using:

```
PRINT POS
```

To print the position of axis 1, the following would be used:

```
PRINT POS.1
```

Note the use of the dot to signify the axis number, see *MINT for NextMove Programmers Manual* for more details on MINT syntax.

The use of *units* may have been noted in the example above. This is because engineering units can be applied to an axis. In a linear table for example, there may be 1000 encoder counts per mm. The keyword **SCALE** (or **SF** for short) can be used to set the new scale factor of the axis:

```
SCALE = 1000
```

This enables positions to be specified in mm and speeds in mm/s. The following shows a more complete MINT example:

```
INTRO.MNT
```

```
REM Program: XY example
REM
```

```

REM Define 10 XY positions
DIM xpos(10) = 10,10,10,20,30,40,40,40,30,20
DIM ypos(10) = 10,20,30,30,30,30,20,10,10,10

GOSUB init
GOSUB main
END

#init
HOME = 0,0      : REM Home both axes
PAUSE IDLE[0,1] : REM Wait for axes to stop
RETURN

#main
REM Repeat forever
LOOP
  REM Move to the 10 points
  FOR a = 1 to 10
    REM Move to the absolute position
    MOVEA = xpos(a),ypos(a) : GO
    PAUSE IDLE[0,1]: REM Wait for axes to stop
    OUT0 = 1      : REM Set an output (head down)
    PAUSE IN0     : REM Wait for head to be down
    OUT0 = 0      : REM Move the head up
    PAUSE IN1     : REM Wait for head to be up
  NEXT
  GOSUB init     : REM Home the axes again
  PAUSE IN1     : REM Wait for input to start again
ENDL
RETURN

```

Although this is a simple example, MINT is equally adapted to more complex applications with keywords for print registration, product infeed, flying shears and cam profiling. MINT structure and examples are described in more detail later in this manual.

The example given would be loaded into the *Program buffer* for execution. The *Program buffer* is used to store the application program for the controller. In addition to the *Program buffer*, there is a *Configuration buffer*. This stores a file usually containing the configuration parameters of the system, such as system gains, speed and accelerations.

A typical *configuration file* for the above example may be:

INTRO.CFG

```

REM INTRO.CFG
REM Baldor Optimised Control

REM Configuration file for XY table using servos

AXES[0,1]      : REM Two axis system

RESETALL      : REM Reset all axes in the system
CONFIG = _servo, _servo

GAIN = 1.5;    : REM Set proportional gain for both axes
KVEL = 10;    : REM Set velocity feedback for both axes

```

```

KVELFF = 0;           : REM Zero feedforward
DGAIN = 0;           : REM Zero derivative gain
KINT = 0;            : REM Zero integral gain
KINTRANGE = 20;     : REM 20%
CURRLIMIT = 100;    : REM 100%

HMINPUT = 0;        : REM Set input 0 as home switch for both axes
ERRORIN = 1         : REM Set input 1 as external error i/p for both axes

SCALE = 400;        : REM Scale factor. Assume units of mm
SPEED = 200;        : REM Speed of 200 mm/sec
ACCEL = 500;        : REM Accel of 500 mm/sec^2
DECEL = 500;        : REM Decel of 500 mm/sec^2
HMSPEED = 50;       : REM Datuming speed
BACKOFF = 10;       : REM Datuming backoff speed factor

MFOLERR = 2;        : REM Max following error of 2mm
IDLE = 0.5;         : REM Idle position of 0.5mm
RAMP = 0;           : REM Trapezoidal
    
```

Program and Configuration buffers are explained in more detail in later sections.

Using the fast interrupt capability on the controller, the positions and encoder values of all axes can be latched within 30 μ s. This latched position can be used as a product reference for product infeed or position verification. In addition to latching the position, a MINT interrupt routine can be called to perform a function associated with the input, immediately.

```

#FASTPOS
  ax1pos = FASTPOS.0
  ax2pos = FASTPOS.1
  OFFSET.0 = ax1pos - x2pos : REM make up the difference
RETURN
    
```

Alternatively the latched encoder values could be used to calculate the required offset.

```

#FASTPOS
  ax1enc = FASTENCODER.0
  ax2enc = FASTENCODER.1
  OFFSET.0 = ax1enc - x2enc : REM make up the difference
RETURN
    
```

Interrupts are placed in the program by simply adding a pre-defined subroutine to the program.

MINT supports a number of interrupt sources in addition to the fast interrupt.

```

#INO
  PAUSE IDLE
  MOVER = 10 : GO
  PAUSE IDLE
RETURN
    
```

This attaches an interrupt to input 0 and will result in a move relative of 10 units in response to a falling edge in the input (assuming the input is configured as NPN, see section 13.2 for hardware details on digital inputs and outputs).

Interrupts have the advantage of working in the background without affecting program flow. Within MINT there are few occasions where program execution is halted. A move can be set-up to operate in the background, at the same time, the program can be checking inputs, writing to outputs and checking position:

```
MOVEA = 100 : GO      : REM Absolute move of 100
PAUSE POS > 50 : REM Wait for position of 50
OUT1 = 1      : REM Set an output
```

This can occur with something more complex; a cam profile for example:

```
CAMTABLE(0, cam0, NULL)
MASTERSOURCE.0 = _msPOS
MASTERCHANNEL.0 = 1
MASTERDISTANCE.0 = 100
CAM.0 = 4
GO.0

JOG.1 = 1      : REM Start motion of master

PAUSE CAMINDEX > 4
OUT1 = 1
```

Cams are a special move type and take the place of mechanical cams. In MINT, a series of positions on the slave (the cam) are referenced against a series of positions on the master. This is given in the form of a table which can then be executed as a one shot or a continuous cam. Cams once executed, operate in the background and give full control back to the program.

In addition to the motion control features, MINT provides sophisticated keywords for controlling terminal output via a VT100 compatible terminal connected to the controller. Keywords are available for locating the cursor on the screen, for printing messages and for formatting input and output. With the CAN Operator Panel, MINT also has the ability to detect when a key is pressed down. This is beneficial for operator jog control.

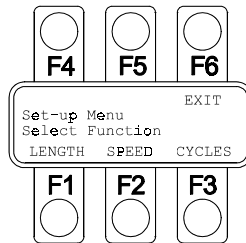


Figure 5: Set-up Menu

4.2 MINT Programmers Toolkit

The CD ROM supplied with the *NextMove* controller is the MINT Programmers Toolkit. This will install:

- MINT application file for the controller
- *NextMove Workbench*
- DOS Utilities
- MINT Interface Library Standard Edition
- Various example programs

There are also some uninstalled files on the CD.

- Various user manuals in .PDF format including :
 - MINT Interface Library Standard and Developer Release Issue 4.2
 - MINT Programmers Manual Issue 3.0
 - NextMove PC WorkBench User's Guide Issue 3.0
 - MINT for NextMove Revision History Issue 3.0
 - cTERM and MINT Utilities for Windows and DOS Issue 3.0
 - CAN Peripherals Installation Manual and Programming Manual Issue 2.0
- A PDF reader
- Disk images for the MINT Programmers Toolkit

These are not installed due to the size of the files concerned.

4.3 NextMove WorkBench

The NextMove *WorkBench* is a selection of tools running under Windows to aid the use of *NextMove PC*. Communication with *NextMove PC* is performed using Dual Port RAM (DPR). DPR is an area of memory that is accessible by both the PC and *NextMove PC*. *NextMove PC* uses DPR to provide the following:

- Pseudo serial port to MINT
- Updating of axis information in real time such as position, speed and error status.
- Update of I/O status in real time such as digital inputs and outputs and analog inputs.

The *WorkBench* provides various tools for interfacing to *NextMove PC* over DPR and to provide an operator interface to any application program running on *NextMove PC*. Tools are included to:

- Examine the contents of any location of DPR.
- Alter the gains of servo axes (at run time if required).
- Examine the position, following error, mode of motion or error of any axes.
- Plot time graphs of axis variables including; position, following error and velocity.

- Examine the state of the digital inputs and outputs.
- Set or clear the digital outputs.
- Examine any analog input.

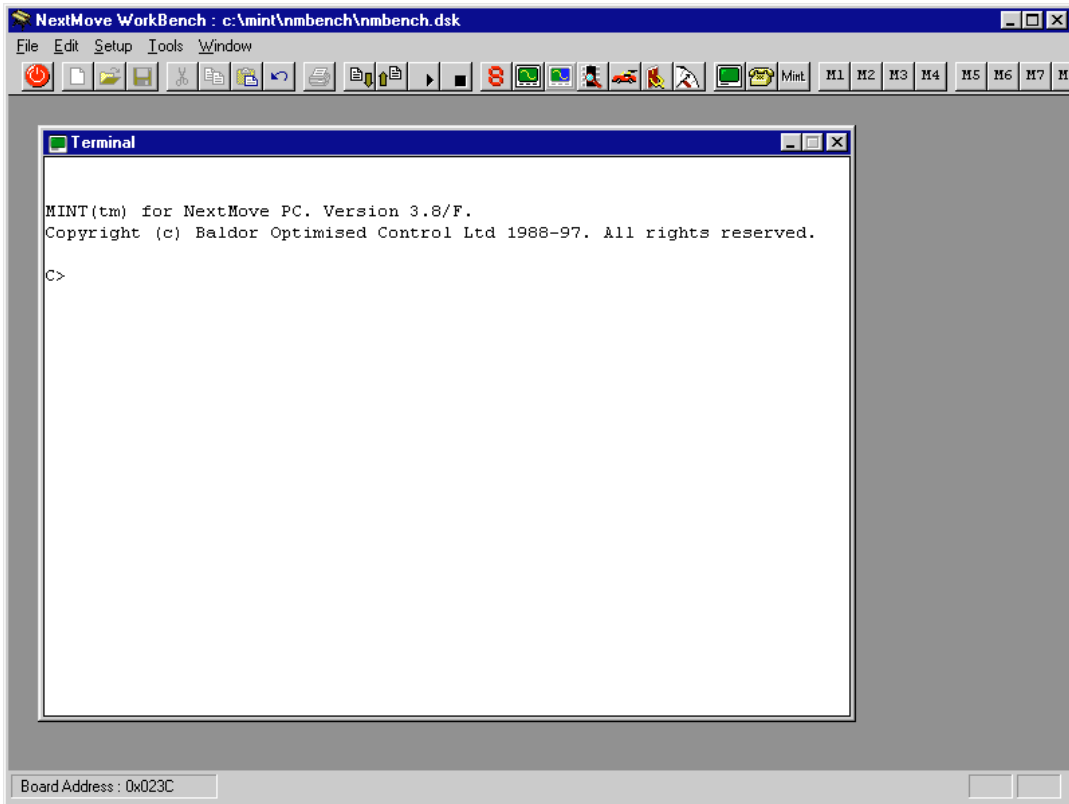


Figure 6: *NextMove WorkBench* Terminal Window

MINT for *NextMove PC* is supported via the following tools:

- A terminal emulator using the pseudo-serial buffer in DPR to allow MINT's on board editor to be used.
- The facility to view the current program error and line being executed.
- An editor with the ability to upload and download to MINT. Separate editors are provided for the program and configuration file.
- Support for the MINT communications protocol allowing data to be passed and read from an executing program.

NextMove WorkBench will be installed during installing of the MINT Programmer's Toolkit.

4.4 Keyword Summary

The following is a complete list (in alphabetical order) of all the keywords supported on *NextMove*. Abbreviations are given following /.

Keyword	Description
<i>Program Constructs</i>	
	: statement separator
	REM comment
<i>Subroutines</i>	
	# subroutine label
	#CAN can event/error subroutine
	#FASTPOS fast position interrupt routine
	#INO . . #IN23 interrupt subroutines on digital inputs
	#ONERROR on error subroutine
	#STOP stop subroutine on stop input
	#TIMER timer event subroutine
<i>Conditional Statements</i>	
	IF DO {ELSE} ENDIF block IF structure
	IF THEN standard IF THEN statement
	PAUSE pause program execution until condition is true
<i>Loops</i>	
	FOR {STEP} NEXT Basic for next loop
	REPEAT . . UNTIL loop until condition is true
	WHILE . . ENDW loop while condition is true
	LOOP . . ENDL loop forever
	EXIT terminate current loop structure
<i>Program Flow</i>	
	AUTO automatic execution of program
	GOSUB branch to subroutine
	GOTO continue execution from label
	END end program execution
	RETURN return execution from subroutine
<i>Terminal Input/Output</i>	
	BEEP issue a beep at the terminal
	BOL place cursor at beginning of line
	CHR output non-ASCII characters
	CLS clear the screen
	ECHO control terminal output
	INKEY / IK read ASCII character from buffer
	INPUT . . USING accept numeric input with optional formatting
	LINE print a string to a specified line
	LOCATE locate cursor on terminal
	PRINT / ? Prints a string or number with optional formatting
	READKEY / RK Read the pressed key on a KeypadNode
	TERMINAL / TM Control terminal output
	VIEW Displays system information

Keyword	Description
<i>Motion Keywords</i>	
ABORT/AB	abort motion - crash stop all axes
ACCEL/AC	set acceleration rate
AUXENCODER/AEN	read auxiliary encoder
AUXENCODERSCALE/AES	set auxiliary encoder scale factor
AUXENCODERVEL/AEV	read auxiliary encoder velocity
AUXENCODERWRAP/AEW	set auxiliary encoder wrapping
BACKOFF/BA	set home speed backoff factor
CAM/CM	Cam profile
CAMAMPLITUDE/CMA	set a scale factor for Cam profile
CAMEND/CE	set Cam end segment
CAMINDEX/CI	read Cam segment number
CAMPHASE	phase shift a Cam profile
CAMPHASESTATUS/CPS	status of current Cam phase
CAMSTART/CS	set Cam start segment
CAMTABLE	set Cam table
CIRCLEA/CA	circular interpolation absolute
CIRCLER/CR	circular interpolation relative
CONTOFF/CO	contouring off
CONTON/CT	contouring on
DECEL/DCL	set deceleration rate
DEMAND/DM	read instantaneous motor demand
DISDRIVE/DS	disable the axis enable output
ENABLE/EB	enable/disable relay output
ENCODER/EN	read encoder
ENCODERSCALE/ENS	set encoder scale factor
ENCODERVEL/ENV	read encoder velocity
ENCODERWRAP/ENW	set encoder wrapping
ENDRIVE/ED	enables the axis enable output
ERRORDECEL/EDC	set the error deceleration rate
FASTAUXENCODER/FAE	read latched auxiliary encoder
FASTENCODER/FEN	read latched encoder
FASTPOS/FP	read latched position
FEEDRATE/FR	slew speed for positional moves
FEEDRATEMODE/FRM	control use of feedrate
FEEDRATEOVERRIDE/FRO	feedrate override percentage
FLY/FY	flying shear
FOLERR/FE	read instantaneous following error
FOLLOW/FL	enable encoder following
FOLLOWMODE/FLM	select type of encoder following
GO	begin synchronised motion
HMSPEED/HS	set speed for home cycle
HOLDTOANALOGUE/HTA	start hold to analog HTA move
HOME/HM	seek home position/read home switch state
HTACHANNEL/HAC	analog channel for HTA move
HTADAMPING/HAD	damping term for HTA move
HTADEADBAND/HDB	deadband for HTA move
HTAFILTER/HAF	analog filter for HTA move
HTAGAIN/HAG	gain term for HTA move
IDLE/ID	return true if idle/set idle following error window

Keyword	Description
INCA/IA	absolute increment move
INCR/IR	relative increment move
JOG/JG	speed control
MASTERCHANNEL/MSC	master axis channel
MASTERDISTANCE/MSD	master axis distance
MASTERSOURCE/MSS	master axis source
MAXIMUMSPEED/MS	sets maximum speed for axis
MODE/MD	return current mode of motion
MOVEA/MA	positional move absolute
MOVER/MR	positional move relative
OFFSET/OF	perform offset during pulse/encoder following
OFFSETDISTANCE/OFD	set a distance over which to perform an offset
OFFSETMODE/OFM	set type of offset
OFFSETSTATUS/OFS	read status of last offset mode
POS/PS	read/set axis position
PRECISIONINCREMENT/PCI	set distance between values in compensation table
PRECISIONMODE/PCM	control action of compensation
PRECISIONOFFSET/PCO	set leadscrew offset
PRECISIONTABLE	load leadscrew compensation table
PREHOMEPOSITION/PHP	axis position prior to end of homing routine
RAMP/RP	velocity profile smoothing factor
SPEED/SP	slew speed for positional moves
SPLINE/SPL	Spline move
SPLINEEND/SPE	Spline end segment
SPLINEINDEX/SPI	read Spline segment number
SPLINETABLE	set up a Spline table
SPLINETIME/SPT	Spline segment duration
SPLINESTART/SPS	Spline start segment
SPLINESUSPENDTIME/SST	Sets the accel/decel time for a pause or controlled termination of a Spline move
STOP/ST	bring axis to a controlled stop
SUSPEND/SSD	bring axis to temporary stop
TORQUE/TQ	set torque control
VECTORA/VA	vector move absolute
VECTORA/VR	vector move relative
VEL/VL	return instantaneous axis velocity
ZERO/ZR	set current axis position as zero
<i>System and Configuration Keywords</i>	
AXES	set the default axis numbers
BACKLASH/BL	sets the size of the backlash
BACKLASHINTERVAL/BLI	sets the rate of backlash compensation
BACKLASHMODE/BLM	sets the mode of backlash compensation
CONFIG/CF	configure axis for off/servo
CURRLIMIT/CL	set current limit of DAC
DACMODE/DCM	configures DAC output
DACMONITORAXIS/DMA	set axis to monitor
DACMONITORGAIN/DMG	set monitor scaling
DACMONITORMODE/DMM	set DAC monitor mode
DEFAULT/DF	return all motion parameters to default values
DEFINE	defines a macro

Keyword	Description
DGAIN/DG	derivative gain term
DIM	reserve space for array data
DRIVETYPE/DT	show possible hardware configuration of axis
FREE	display free memory
GAIN/GN	servo loop proportional gain
GEARING/GR	gearing compensation %
GEARINGMODE/GRM	gearing compensation mode
KACCEL/KA	servo loop acceleration feed forward gain
KINT/KI	servo loop integral gain
KINTMODE/KIM	set how integral gain is used
KINTRANGE/KR	servo loop integration range
KVEL/KV	servo loop velocity gain
KVELFF/KF	servo loop velocity feed forward gain
LASTMOVEBUFFERID/LBI	read the move identifier of the previous move
MBFREE/MBF	read free space in move buffer
MOVEBUFFER/MB	set or return size of move buffer
MOVEBUFFERID/MBI	set or read the move identifier
MOVEBUFFERSTATUS/MBS	read the status of the move buffer
LOOPTIME/LT	set servo loop time
MFOLETT/ME	set maximum following error
NUMBEROF/NO	displays number of hardware resources
PRESCALE/PR	pre scale encoder counts
RELEASE	clear user variables from memory
RESET/RE	re-initialise motion variables
RESETALL/RA	RESET all axes
RUN	execute a program
SCALE/SF	scale encoder counts to user units
TIME/TE	user timer
TIMEREVENT/TEV	Start a user timer event
TROFF	program trace off
TRON	program trace on
VER	display MINT version number
WAIT/WT	wait in milliseconds
ZLEVEL/ZL	set encoder Z pulse level
 <i>Event, Error and Safety Feature Keywords</i>	
ADCERROR/AE	read analog channels in error
AXISSTATUS/AS	returns current warning status for axis
CANCEL/CN	clear error/cancel current move
DINT	disable interrupts
DISLIMIT/DL	disable hardware/software limits switches
EINT	enable interrupts
ENLIMIT/EL	enable hardware/software limit switches
ENOUTPUT/EO	assign an output as a drive enable
ERL	line number on which error occurred
ERR	returns error code
ERRAXIS	axis number of error code
ERROR/ER	return axis error condition
ERRORIN/EI	assign an input as external error input channel
ERRORMASK/EM	controls error handling
FEMODE/FM	control following error detection
FWLIMITIN/FLI	assign input as forward limit input

Keyword	Description
FWSOFTLIMIT/FSL HMINPUT/HI HWLIMIT/HL IMASK/IM IPEND/IP LASTERR MAXANALOGUE/MXA MINANALOGUE/MNA MONCHANNELS/MCL REVLIMITIN/RLI REVSOFTLIMIT/RSL SOFTLIMIT/SL STOPINPUT/SI STOPMODE/SM	set forward software position limit assign an input as a home input set action taken on a hardware limit input interrupt mask on user inputs (#INx) user interrupts pending display last interpreter error message upper analog limit value lower analog limit value assign analog channels to an axis assign input as reverse limit input set reverse software position limit sets action taken on a soft (position) limit assign input as an axis stop input sets action taken on a stop input
<i>Input/Output Keywords</i>	
ACTIVEINLEVEL/AII ACTIVEOUTLEVEL/AOL ADCMODE ANALOGUEx/Ax DAC/DC IN INx/Ix INACTIVEMODE/IAM LIMIT/LM LIMITF/LF LIMITR/LR NEGEDGEIN/NEI OUT/OT OUTx/Ox POSEDGEIN/PEI SERIALBAUD/SB SERIALNODE/SND STOPSW/SS	sets or returns active high inputs sets outputs to be active high when on sets the mode for the analog input channel read analog inputs (0 to 7) write direct to 14 bit DAC output read 16 bit value of digital inputs read digital inputs 0 to 23 sets edge or level triggering on digital inputs read limit switch input values read state of forward hardware limit read state of reverse hardware limit sets negative edge triggered inputs write value to digital output write to user outputs bits 0 to 11 sets positive edge triggered inputs set the baud rate of the serial ports set the node number for RS485 multi-drop communications read stop switch status
<i>CAN Control</i>	
CANBAUD/CB CANSTATUS/CST NODELIVE/NL NODETYPE/NT KEYPADNODE/KN READKEY/RK REMOTEBAUD/RB REMOTEBOUNCE/RD REMOTESETUP REMOTESTOP/RES REMOTEIN/RI REMOTEINPUTACTIVELEVEL/RIA REMOTENODE/RN REMOTEOUT/RO	set CAN baud rate read CAN status read node status add node to CAN bus specify i/o channel for keypad node read ASCII code of current key press on keypad node set node baud rate CAN input debounce time set the node number and baud rate of a CAN node Control emergency stop state read input node set node input active level assign node number set output node

Keyword	Description
REMOTEOUTPUTERROR/ROE	read/clear output errors
REMOTEOUTPUTACTIVELEVEL/ROA	set node output active level
REMOTERESET/RR	reset a CAN node
REMOTESTATUS/RS	read /clear remote status
STATUSNODE/SN	read node number in error
<i>Dual Port Ram Functions</i>	
PEEK/PK	read word from DPR
POKE	write word to DPR
<i>Comms Functions</i>	
COMMS	read/write value using protected communications mode
COMMSMODE/CMM	enable RS232/RS485 multi-drop comms
<i>Data Capture</i>	
CAPURE/CP	start/stop data capture
CAPTUREAXIS/CPA	set axis from which to capture data
CAPTUREMODE/CPM	set quantity to capture
CAPTUREINTERVAL/CPI	set sample interval
<i>Line Editor Commands:</i>	
CON	edit configuration file
DEL	delete lines from a program
EDIT	edit a program line
INS	insert lines into program
LIST	list a program
LOAD	load a file
NEW	clear program from memory
PROG	edit program file
SAVE	save a file
<i>Relational and Mathematical Operators</i>	
+ - / * < > <= >= <> =	
~	bitwise toggle
ABS	return absolute value of number
ACOS	returns arc-cosine
AND/&	bitwise AND
ASIN	returns arc-sine
ATAN	returns arc-tangent
COS	returns cosine of angle
EXP	exponential function
INT	returns integer of number
LOG	returns natural logarithm of number
MOD/%	modula arithmetic operator
NOT/!	Logical not
OR/ 	bitwise OR
SIN	returns sine of angle
SQRT	returns square root of number
TAN	returns tangent of angle
XOR	exclusive OR

5. Options

NextMove PC supports a number of options from CAN bus I/O expansion boards to program development tools using the *MINT Interface Library*.

Order codes are given in the following table. Also refer to Section 10 for configuring *NextMove PC* with CAN Peripherals.

Ordering Information:

Item	Order Code	Description
Options:		
CAN Operator Panel	KPD002-502	CAN bus based 27 key keypad and 4 line LCD display
InputNode 8	ION001-503	CAN bus 8 digital inputs
OutputNode 8	ION003-503	CAN bus 8 digital outputs
RelayNode 8	ION002-503	CAN bus 8 relay outputs
ioNode 24/24	ION004-503	24 inputs and 24 outputs
Axis Expansion Card	OPT009-501	Allows 8 axes of servo and 8 axes of stepper.
Axis Expansion Card	OPT011-501	Allows 8 axes of stepper.
Encoder splitter	OPT008-501	Provides two buffered outputs from one encoder input.
Breakout Board	NMP004-501	Provides easy access to the 100 way connector via screw down terminals.
Breakout Cable 1.0 m	CBL005-501	1.0 m 100 way cable to attach <i>NextMove PC</i> to Breakout board.
Breakout Cable 1.5 m	CBL005-502	1.5 m 100 way cable to attach <i>NextMove PC</i> to Breakout board
Breakout Cable 3.0 m	CBL005-503	3.0 m 100 way cable to attach <i>NextMove PC</i> to Breakout board
Miscellaneous:		
<i>NextMove PC</i> manual	MN1257	Includes software
MINT Interface Library	SW1259	Libraries for Windows developers

These options are described in more detail in the following sections.

5.1 CAN Operator Panel (*KeypadNode*)

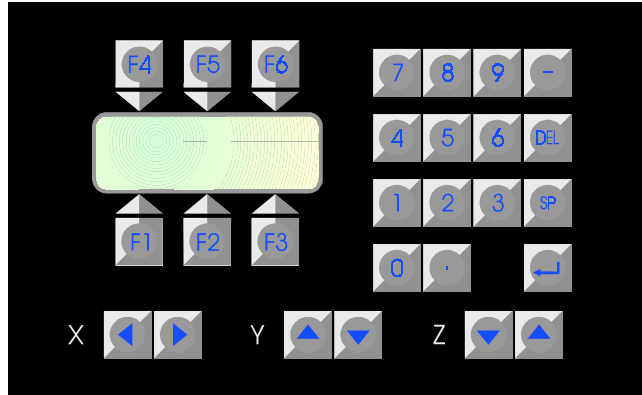


Figure 7: KeypadNode

The CAN Operator Panel (*KeypadNode*) provides a general purpose operator panel suitable for stand-alone machines of all types. *KeypadNode* is cost effective for simple functions, such as replacing thumb wheel switches and providing simple diagnostics. It may also be used as a fully interactive programming panel for special purpose machine control.

KeypadNode operates over the CAN bus, providing noise immunity for distances typically up to 500m (approx. 1640ft).

A range of keywords are available in MINT which allow the easy manipulation of display and keypad. Formatted numeric input and output can easily be achieved using the **PRINT . . USING** and **INPUT . . USING** keywords.

5.2 CAN I/O Nodes

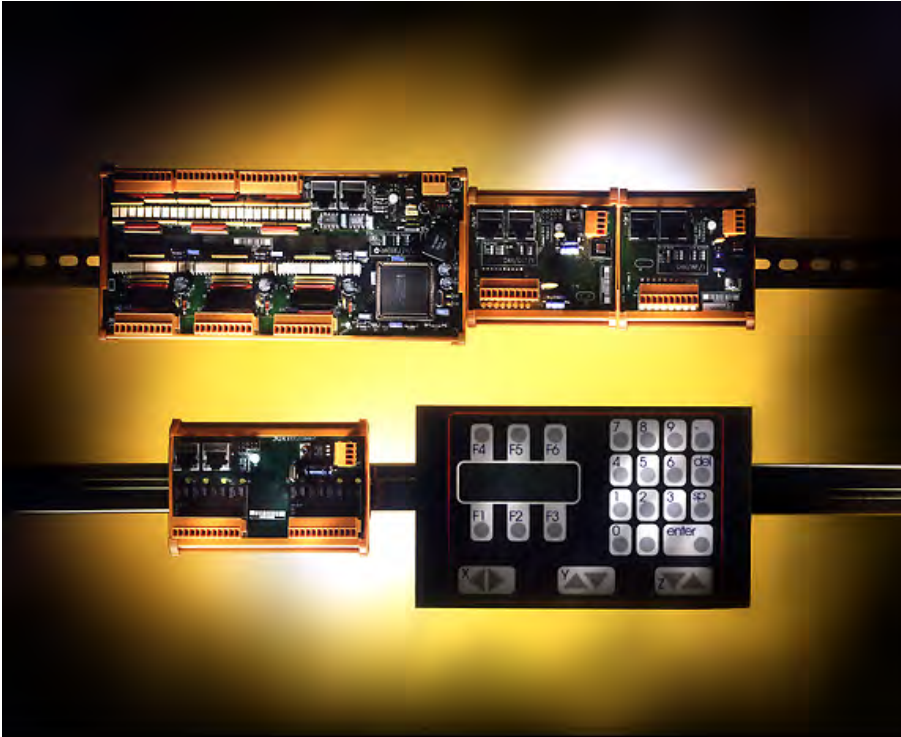


Figure 8: *ioNODE* Product Family

Digital I/O can be expanded easily on *NextMove PC* using the *CAN bus interface*. This provides a high speed and secure serial bus interface to a range of I/O devices as described:

- *InputNode 8*: 8 opto isolated digital inputs.
- *OutputNode 8*: 8 opto isolated digital outputs with short circuit and over current protection.
- *RelayNode 8*: 8 relay outputs.
- *ioNode 24/24*: 24 input and outputs

These I/O devices, known generically as *ioNODE*, operate on the same bus as *KeypadNode*, the CAN bus operator panel. Refer to Section 10 for further information on CAN Peripherals.

5.3 The Encoder Splitter/Buffer Board

This is a stand alone PCB that takes an encoder signal, either single ended or differential and gives 2 differential outputs. This is useful for 'daisy chaining' an encoder signal from a master across a number of controllers.

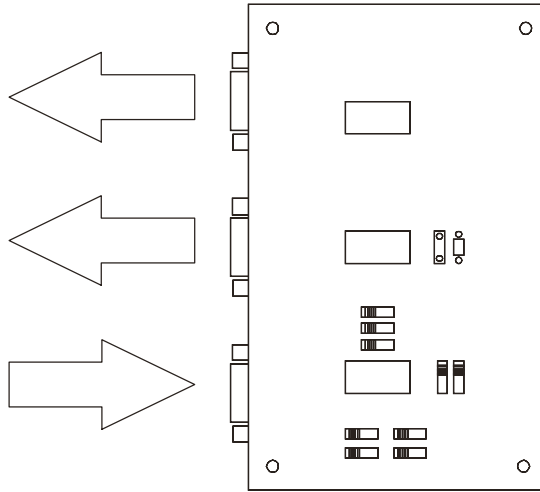


Figure 9: Encoder Splitter/Buffer Board

5.4 Expansion Boards

Two box headers on the *NextMove PC* controller provide an expansion bus for the controller. There are two boards available which are directly supported by *NextMove PC*, both of which expand the axis capability of the controller. Refer to Section 8.11.

5.5 Breakout Board

The NextMove PC break-out board is designed to give easy access to the various I/O and motor control signals.

The break-out module takes the *NextMove PC* signals on the 100-way D-type connector and routes them to screw-down terminals for the digital I/O, 9-way D-types for the encoders and RJ-45s for the CAN.

For more details see section 13.

5.6 MINT Interface Library

The *MINT Interface Library* is a CD-ROM containing a set of libraries for host applications on the PC. The MINT Interface Library supports all *Baldor Optimised Control* products. The following features are supported in *NextMove PC* :

- Ability to upload and download MINT programs and configurations to the controller from a host application.
- The MINT Comms Protocol, allowing a host application to send data to an executing MINT program.
- Immediate Command Mode (ICM) enables a host application to send a motion command to the *NextMove PC*

Support is provided for all versions of Windows (Windows 3.11, Windows 95 and Windows NT) allowing 16 and 32 bit applications to be written. With a Dynamic Link Library (DLL) interface, any programming language supporting DLLs can also support the MINT Interface Library. Interfaces have been supplied for the following languages:

- Borland Delphi
- Microsoft Visual Basic
- Microsoft Visual C++
- Borland C++
- C

Contact a local sales office for details.

6. Getting Started

This section is step by step guide to setting up a basic *NextMove PC* servo and stepper control system. This section assumes very basic familiarity with PC's and the Windows environment.

Before setting up *NextMove PC*, the following items required for getting started should be made available:

1. This *NextMove PC* Installation Manual.
2. A PC running Windows 3.1 or 95 with at least one free ISA slot.
3. *NextMove WorkBench* installed on the PC. *WorkBench* is found on the CD-ROM attached with this manual named MINT Programmer's Toolkit. Refer to section 10 for details on installing *WorkBench*.
4. The *NextMove PC* controller, breakout board and 100 pin *Nexmove PC* to breakout board cable.
5. The servomotor/drive combination that is intended for use.
6. A small screwdriver (less than 3.5mm (1/8") blade), soldering equipment and some electrical cable.
7. The appropriate controller, amplifier cables, or the connectors in order for them to be constructed.

The system to be configured will consist of one axis of servo motor and one axis of stepper motor. If the configuration is not exactly as described, it is still possible to proceed since much of this guide remains relevant.

It is worthwhile but not essential to be familiar with the MINT programming language and *NextMove WorkBench* before starting this set-up procedure. A summary of the use of MINT is given in section 7 of this manual and covered in more detail in the *MINT for NextMove Programmers Manual*. Refer to section 11 for information on the MINT Programmers toolkit and the *NextMove Workbench*.

During installation of *NextMove PC*, it is advisable to record the system setup parameters.

6.1 Installing the *NextMove PC* Controller

The *NextMove PC* card is a standard full-length, full height PC-AT plug-in card. The nominal overall dimensions are 339mm (13.27") long by 122mm (4.8") deep. Before purchasing the host machine, it should be checked that this size of card can be accommodated. Some small desktop personal computers restrict the maximum depth of a plug-in card to 106mm.

The host must have a spare **full length** card slot. It must also be an AT type machine as this card will not work correctly in a PC-XT. The card dimensions conform to the ISA standard except in respect of the 100 way D-type connector on the metal bracket, which hangs below the bottom edge of the card and is almost level with the ends of the gold fingers. However, this is unlikely to cause any difficulty in fitting a card.

NextMove PC draws approximately 830mA @ +5V, 166mA @ +12V, 217mA from -12V and 0mA from -5V. The exact figures depend on the loading.

The PC communicates with the *NextMove PC* via two contiguous 16 bit I/O locations, these are selectable using DIP switch S1 on the *NextMove PC*. The first address accesses a 16 bit address latch, the second accesses the dual port RAM.

Before inserting the *NextMove PC* it is important to set it up to use a free I/O address space. The positions of the four switches (S1) and jumper H modify the address of the *NextMove PC*.

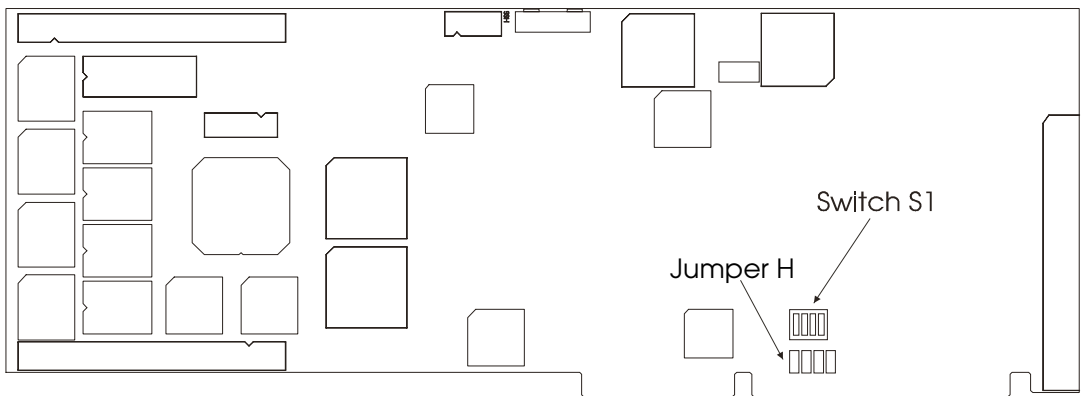


Figure 10: Location of S1 and Jumper H

The possible address locations of the *NextMove PC* are listed below.

DIP switch S1:- 1..4	Address Register (Hex)	Data Latch (Hex)
on,on,on,on	300	302
on,on,on,off	304	306
on,on,off,on	308	30A
on,on,off,off	30C	30E
on,off,on,on	310	312
on,off,on,off	314	316
on,off,off,on	318	31A
on,off,off,off	31C	31E
off,on,on,on	320	322
off,on,on,off	324	326
off,on,off,on	328	32A
off,on,off,off	32C	32E
off,off,on,on	330	332
off,off,on,off	334	336
off,off,off,on	338	33A
off,off,off,off	33C	33E

If jumper H (located beneath the DIP switch) is fitted then 100 (hex) should be subtracted from the above address values.

Note: *NextMove PC* is accessed using 16 bit cycles only. However it will react to 8 bit cycles to allow “Plug’n’Play” BIOS systems to recognise it as a legacy card.

The I/O locations selected to communicate with the NextMove PC must not already be in use by another device. This will cause a conflict within the I/O space and it is likely that neither device using the conflicting I/O will function correctly.

It is possible to determine the use of I/O address space within the PC by the following procedure in Windows 95:

1. Click the windows *Start* button and choose *Settings* ⇒ *Control Panel*
2. From the *Control Panel*, double click on the *System* icon.
3. From *System Properties* click the *Device Manager* tab.

From the list of icons displayed, click the *Computer* icon (first in list) and click the *Properties* button at the bottom of the window and ensure that *Input/output (I/O)* is selected.

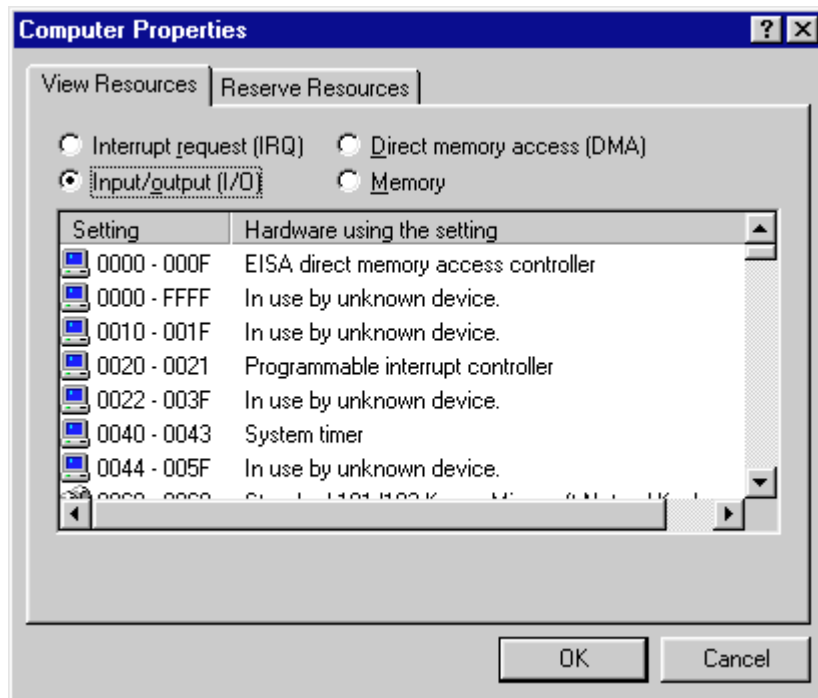


Figure 11: Windows '95 I/O map

Various interrupt levels are jumper selectable, this is not required unless using a DOS based host application to handle interrupts from the card. Details on setting the interrupt level on the NextMove PC can be found in section xx.

6.1.1 Before Inserting the Card

Before placing the card in the computer, carry out the following steps.

1. Exit any applications that are running and close all windows.
2. Turn off the power and unplug all power cords.
3. Remove the cover from your computers system unit.

Note that before touching the card, be sure to discharge static electricity from hands by touching a grounded metal surface, such as the screw on an electrical outlet's plate cover.

6.1.2 Placing the Card In the Computer

Locate an unused 16 bit, full length slot.

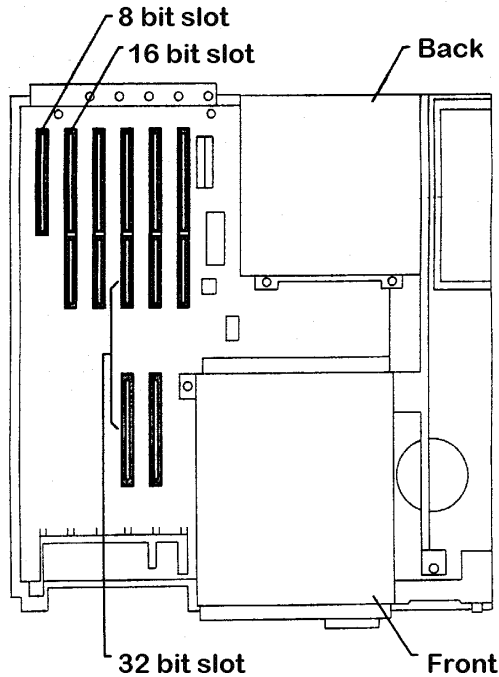


Figure 12: PC Slots

This illustration, looking down at a computer, shows how slots may look in the PC. The computer may look different, especially if it is an EISA or PCI type.

- Remove the cover from the slot, and save the screw for use later.
- Discharge any static electricity from hands.
- Remove the card from its protective wrapper. Do not touch the gold contacts at the bottom of the card.
- Align the bottom of the card (gold contacts) with the slot and press the card firmly into the socket. When correctly installed, the card locks into place.
- Make sure that the top of the card is level (not slanted) and that the hole on top of the card's metal bracket lines up with the screw hole at the back of the slot.
- Anchor the card in place using the screw removed earlier.
- Replace the computer cover and screws.
- Reconnect any cables and power cords that were disconnected or unplugged.

6.2 Installing The Software

The software is supplied on CD-ROM. To install the software insert the CD-ROM into the PC. Under Windows '95 this action initiates the automatic loading of a set-up wizard. Under Windows 3.1 run the executable SETUP.EXE in the root directory of the CD-ROM this will run the set-up wizard. This Wizard allows either a complete installation of all the available software options, or a customized set-up. The on-screen instructions should be followed.

The Set-up Wizard copies the *WorkBench* files to appropriate directories on the hard disk. There is a choice of what drive and directory to use. The default directory is **C:\MINT**.

6.2.1 Starting The WorkBench

To start *WorkBench* from the Windows 95 *Start* menu choose:

1. *Programs*
2. *MINT Tools*
3. *NextMove PC WorkBench*

To start *WorkBench* from the Windows 3.x:

1. Open the *MINT Tools* program group
2. Double click on the *NextMove PC WorkBench* icon

The following section describes the two basic steps necessary to communicate with the *NextMove PC* controller.

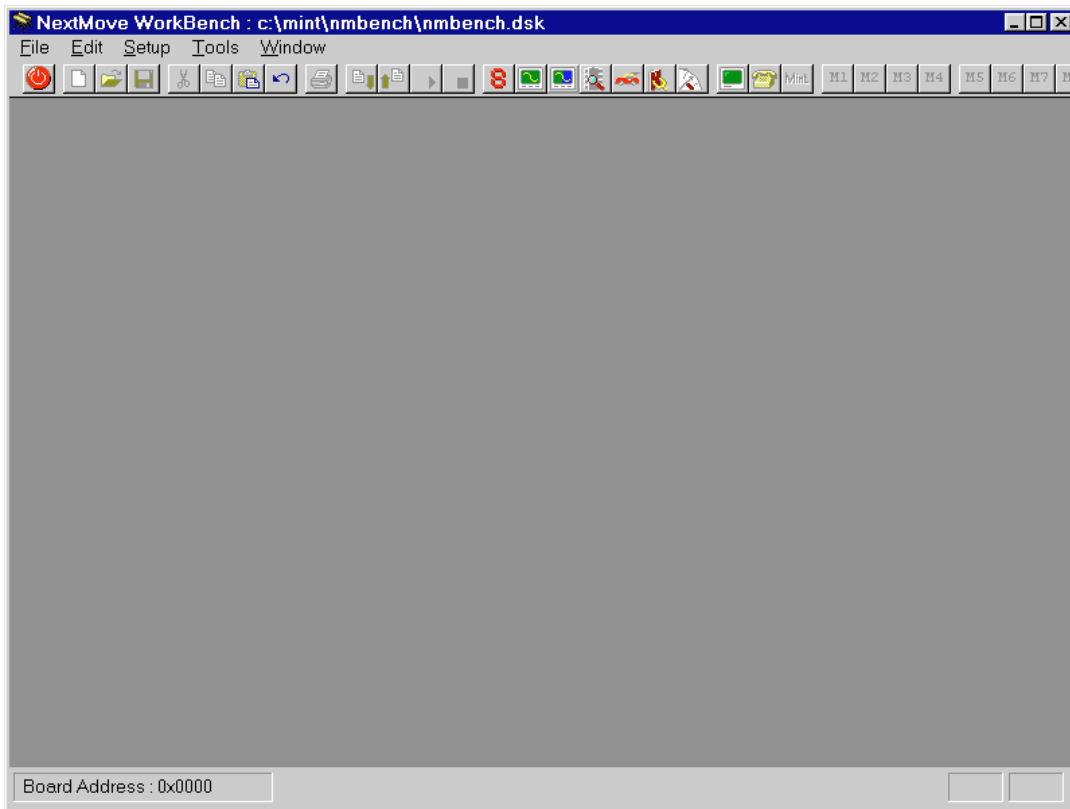


Figure 13: WorkBench Default Window

6.2.2 Selecting A NextMove PC

The *WorkBench* only supports one *NextMove PC* at a time although multiple instances of the *WorkBench* can be run. The address of the *NextMove PC* must be supplied before any of the tools can be used. To tell the *WorkBench* which address *NextMove PC* is using, either:

- Click on the *Board Address* label at the bottom of the *WorkBench* application or
- From the *Setup* menu select *Board Address*.

The address can be entered by selecting from the list of valid (hexadecimal) addresses by clicking on the *down* arrow adjacent to *NextMove Address (Hex)*.



Figure 14: NextMove Address Window

Alternatively the *NextMove WorkBench* can automatically detect the *NextMove PC* using the *Auto Search* function. This writes to all the possible *NextMove PC* I/O address locations and checks for an appropriate response.

As the auto search function writes to all *NextMove PC* I/O locations the functionality of any other devices that use these I/O locations may be affected by this function. This may cause the PC to lock up.

To confirm that the *NextMove WorkBench* is communicating with the *NextMove PC* correctly click on the *Test* and *Reset* buttons, both of which should be successful.

Once an address has been selected, the *WorkBench* will enable any menu items supported by the board at that address. If no application is currently running on *NextMove PC*, only the menu option *Download Application* in the *File* menu will be highlighted. It will be necessary to download an application to *NextMove PC*.

6.2.3 Downloading an Application.

An *Application* is a program running on *NextMove PC* to give it its functionality in much the same way as a **.EXE** file on a PC. The application is not to be confused with a MINT *program*. The applications provided is **MINT.OUT**, this is a version of MINT that executes on *NextMove PC*. This includes the ability to call MINT Motion Library functions from the host using the MINT Interface library.

To download an application to *NextMove PC*:

Select *Download Application* from the *File* menu. This will show the *Download Application* dialogue box. Select the file **MINT .OUT** and then click OK.

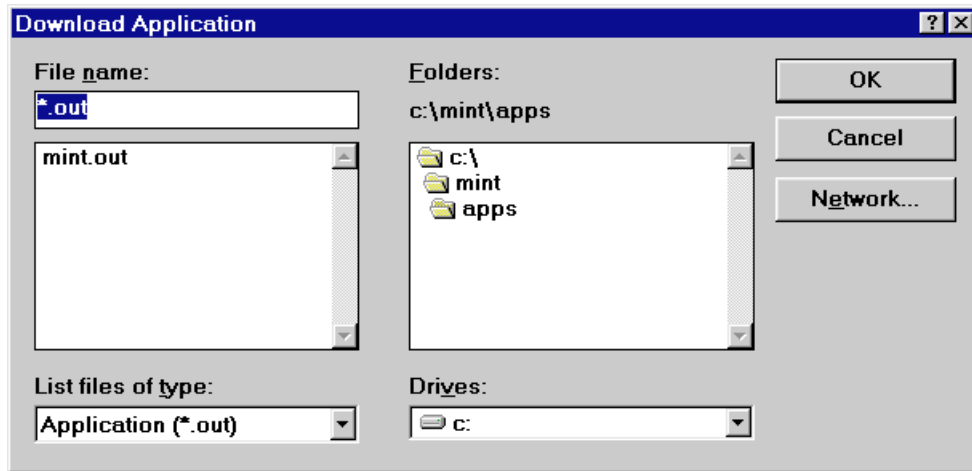



Figure 15: Download Application

When an application is downloaded to *NextMove PC*, the *WorkBench* will determine which tools are supported by the application. Any non-supported tools that are currently running on the *WorkBench* will be closed down.

Once an application file has been successfully downloaded access to the MINT command line is gained via the Terminal window. To enable the Terminal window either select *Terminal* from the *Tools* menu or click the  icon.

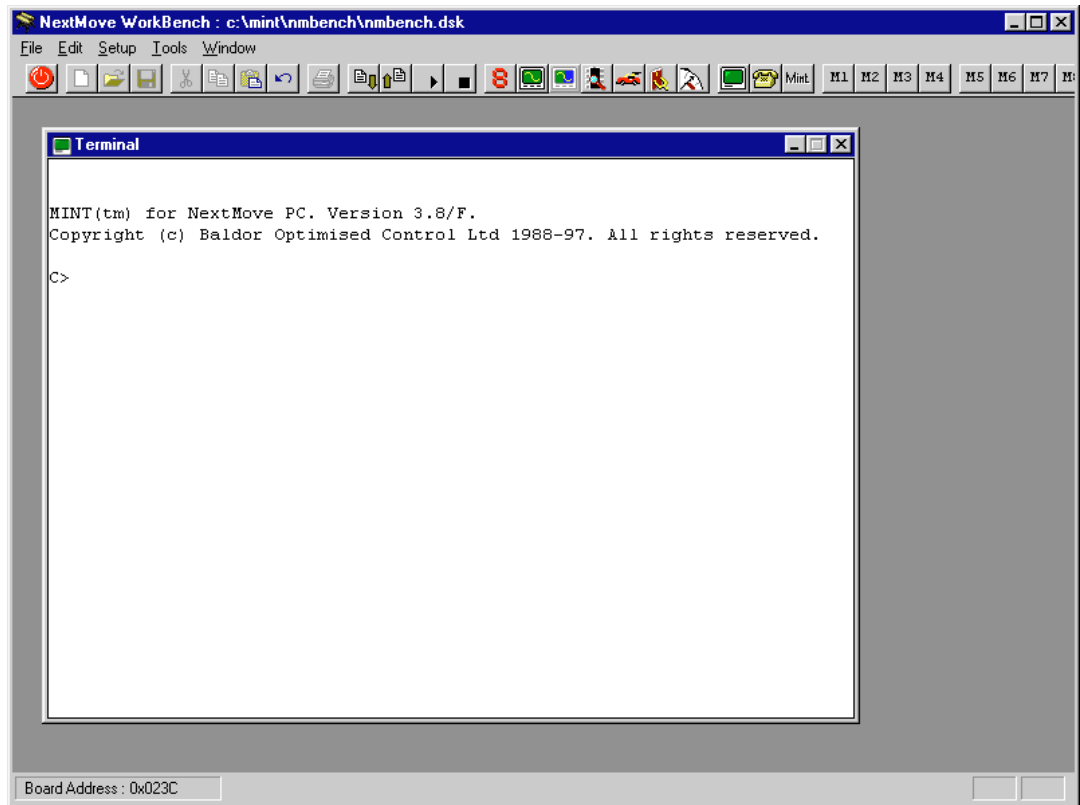


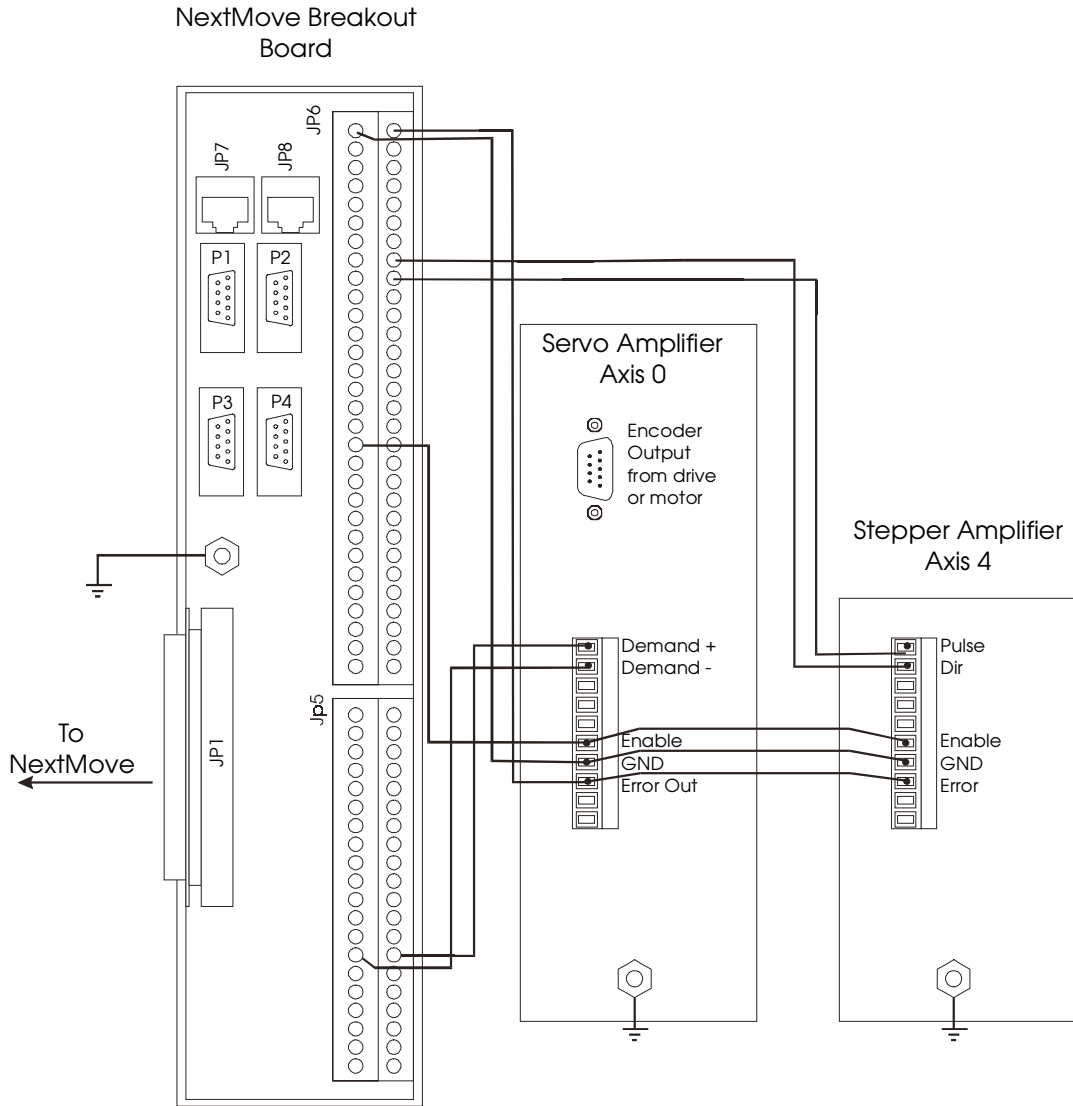
Figure 16: Terminal window

6.3 Minimum System Wiring Example

The following section is a guide to setting up a minimum system. A minimum system is one where the controller and drives are configured to work with as little external wiring as possible. It is recommended that motors are tested and commissioned ‘on the bench’ and not within the machine. This step-by-step example covers setting-up a system with one servo axis and one stepper axis. On the *NextMove PC* controller there are four axes of servo motor control (called 0, 1, 2 and 3 when programming in MINT) and four axes of stepper motor control (called 4, 5, 6 and 7 when programming in MINT).

Connections to the controller are normally made via a cable assembly and break-out board (supplied as an option). The board takes the *NextMove PC* signals on the 100-pin ‘D’-type connector and routes them to screw-down terminals for the digital I/O, 9-pin ‘D’-types for the encoders and RJ-45’s for the CAN. Details of connections to the break-out board can be found in Section 13.

The following figure is a simplified wiring diagram for the two axis system mentioned above. It should be noted that this is not the only possible configuration. It is important to read the associated text before attempting the set-up.



The following table shows the minimum pin connections:

Break-out Board Pin	Name of Signal	Function	Connection on Drive
JP5: 13	demand0	Demand signal for axis 0	demand+/command+ input
JP5: 14	agnd	Reference for analog signals	demand-/command- input
JP6: 26	dig_in_16	Error input (Default setting)	error output
JP6: 37	pulse0/2	Pulse output for stepper motor axis 4	pulse input
JP6: 39	dir1/3	Direction output for stepper motor axis 4	direction input
JP6: 53	relay-com	Common connection of relay	enable input
JP6: 54	relay-no	Normally open connection of relay	Amplifier/Digital Ground

Details on the *NextMove PC* breakout board can be found in section 13.

6.4 Setting Up A Servo Drive

A typical closed loop positioning system can be broken down into three elements:

1. **Position controller** - performs real time positional control of the motor(s), stores the application program and communicates with the user and other control equipment.
2. **Servo amplifier** - takes demand signals from the position controller to control the torque or speed of the motor/actuator.
3. **Motor/actuator** - translates electrical power from the servo amplifier into rotary or linear movement. The motor is fitted with a position sensor which feeds the output position back to the controller.

The controller works by sampling the position of the motor at regular intervals and comparing this position with its target position. It then instructs the amplifier to drive the motor to correct any positional error. This process is repeated typically 1000 times per second to ensure that the motor is always in the correct position. A typical closed loop control system is illustrated in Figure 18.

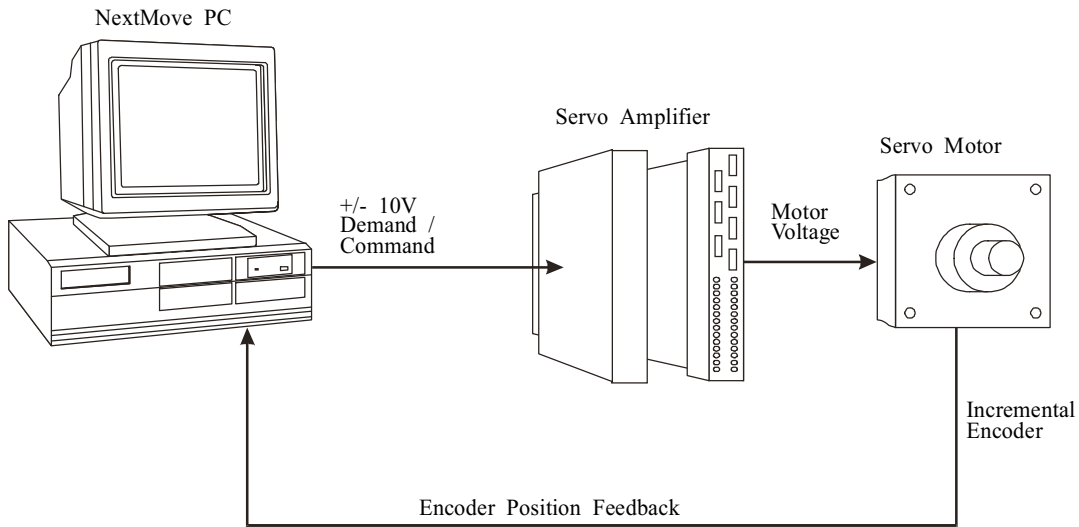


Figure 18: Typical closed loop positioning system

6.4.1 Ground Connections

A good ground connection to the controller is essential for noise immunity in industrial environments. Bad ground connections can be the cause of many strange problems such as loss of motor position.

- ☞ Connect the ground connections stud on the amplifier to the ground connections stud on the break-out board using heavy duty cable. Connect this stud to a common ground connections point using heavy duty cable. See Figure 17.

6.4.2 Encoder Connections

The encoders are the position sensors used by the controller to measure axis position. They consist of two pulse trains, 90 degrees out of phase. The controller uses the phase difference to determine direction of motion and counts the encoder edges to determine position. The frequency of these counted edges reflects the motor velocity.

There are four encoder connectors on the break-out board numbered as follows:

Encoder	Connector
0	P1
1	P2
2	P3
3	P4

The encoder inputs are brought out onto 9 pin 'D' type female sockets.

The controller will work with three channel incremental encoders (**chA**, **chB**, **index**) and with both single ended TTL or differential line driver TTL output types. It is recommended that line driver outputs be used in all applications, since this gives increased noise immunity. Maximum cable length is dependent on the encoder specification, but should be kept as short as possible (details can be found in section 8.5).

It is important that each encoder cable is screened independently and that the screen/shield is connected at the controller end only.

In many brushless amplifiers, the motor is fitted with a resolver and the encoder signal is synthesised by the amplifier. In these instances, the encoder connections are wired to the amplifier rather than the motor.

When making connections to encoder outputs from a brushless drive, do not connect the +5V supplies on the controller and amplifier together since this may cause noise problems.

The encoder should be wired to a 9 pin 'D' male plug, using good quality multi-conductor screened/shielded cable, according to the diagram shown in section 13.1. If the encoder is a single-ended type (i.e. no complement outputs) leave the **!chA**, **!chB** and **!index** pins unconnected. If the encoder does not have an index output, leave the **index** and **!index** unconnected.

The encoders are connected via 9 pin ‘D’ connectors on the breakout board (see section 13.1 for details).

6.4.3 Amplifier Demand/Command Signals

The controller uses +/-10V outputs as a **demand/command** voltage for the amplifier. This **demand** signal relates to a speed or torque demand for the motor. The speed or torque range is determined by the amplifier set-up.

Before proceeding further with the set-up procedure, please ensure the amplifiers have been correctly set-up for the motor in accordance with the instructions in the amplifier manual.

There are four **demand/command** outputs on the *NextMove PC* called **demand 0** to **demand 3**. These correspond to axes 0 to 3 respectively. They can be found on the breakout board connector J5.

For the single axis system connect the axis 0 **demand+/command+** to **demand 0** (pin 1) and **demand-/command-** to **agnd** (pin 2).

6.4.4 Amplifier Enable

The **enable** output from the controller is a relay where **relay_com** is the common connection, **relay_nc** is normally closed and **relay_no** is normally open.

It is essential that the amplifiers are disabled during power-up of the controller or when a controller error occurs. This ensures that the amplifiers are only able to drive the motors when the controller is performing servo-loop closure.

The actual connections that are required will differ for the particular configuration of the **enable** input of the amplifier. Two common examples are given below:

1. If the amplifier enable input must be grounded to enable the amplifier:
Connect the common terminal **relay_com** to the enable input and the normally open output **relay_no** to the amplifier ground.
2. If the amplifier enable input must be tied to 24V to enable the amplifier and grounded to disable the amplifier:
Connect the common terminal **relay_com** to the **enable** input on the amplifier, the normally open output **relay_no** to +24V and the normally closed output **relay_nc** to the amplifier ground.

6.4.5 Limit And Stop Switches

For the purposes setting up the controller, these inputs can be ignored. This is because the software is initially set-up to ignore all errors, (except a following error). For information on configuring these inputs, please refer to section 6.6.

6.4.6 Testing System Wiring

In order to check that the encoder and motor are wired up correctly, it is recommended that the motor is tested and commissioned ‘on the bench’ and not when installed in the machine.

In order to verify that the system is wired up correctly, the following steps should be performed:

- Check that enable output (relay) is the correct sense.
- Check that the encoders works.
- Check that the amplifier is working.
- Check that the encoder and motor are connected correctly.

6.4.7 Checking The Encoder

The encoder records the position of the motor in a positive and negative direction. The following procedure (using the *WorkBench*) can be used to check that the encoder is functioning correctly:

First ensure that the *WorkBench* is running and communicating with the controller and the MINT application is running. All gains must be set to zero and the amplifiers disabled so that the motor shaft can be turned by hand. Then, by moving the motor backwards and forwards the position reading should change.

- From the *Set-up* menu choose *Gains*.
- From the *Tool* menu choose *Motion Watch Window*.

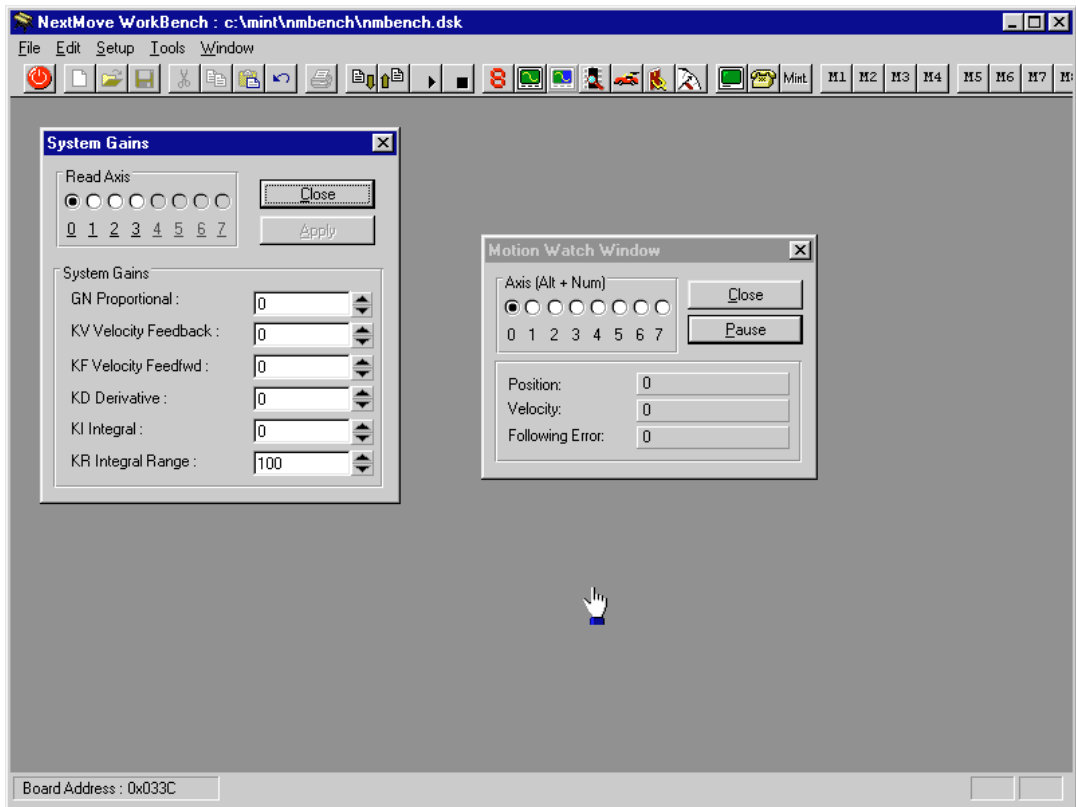


Figure 19: Gains

Ensure the gains **GN**, **KV**, **KF**, **KD** and **KI** are set to zero. Update and Close the window.

Choose the axis whose encoder position is to be viewed and then move the motor shaft by hand. By moving the motor backwards and forwards it is possible to see the position reading change. Note that the controller uses quadrature decoding which gives 4 counts for each line of the encoder disk. With a 1024 line encoder, the position should change by ± 4096 for every revolution moved clockwise or anti-clockwise. If the position does not change then check the following:

- Axis 0 encoder cable is connected to encoder input **0**.
- The encoder has power.
- The encoder is correctly wired up.

Make a note of which direction gives an increase in position. Repeat for other axes.

6.4.8 Checking the Drive Enable

The **drive enable** relay allows *NextMove PC* to shut down the drive in the event of an error. On power-up, the **drive enable** output will be disabled until a command such as **RESETALL** is encountered within the *Configuration* or *Program buffer*. In order to check that the drive enable relay is correctly wired up, the drive should firstly be enabled with:

➤ **RESETALL**

The drive is now enabled. An error can be simulated and the drive disabled with:

➤ **ABORT**

The drive should now be disabled. If this is not the case, then the wiring and drive set-up should be checked.

6.4.9 Checking Motor Polarity

To confirm the polarity of the motor connections a MINT program is used to send commands to the controller. The program is called **MOTOR.MNT** and can be found in the directory **MINT** on the disk supplied with the controller.

MOTOR.MNT

```
REM MOTOR.MNT
REM For 4 axis controller
REM Baldor Optimised Control

RESETALL
TORQUE[1,2,3] = 0; : REM set the DAC for axes 1, 2 and 3 to zero
TORQUE.0 = 1      : REM Increase until motor moves
```

File is in the MINT\NMPC\EXAMPLES directory.

To download the program to the controller:

- From the *File* menu select *Open File* and *Program*.

Find the program **MOTOR.MNT**. Opening this program will place it into a *Program Editor* window.

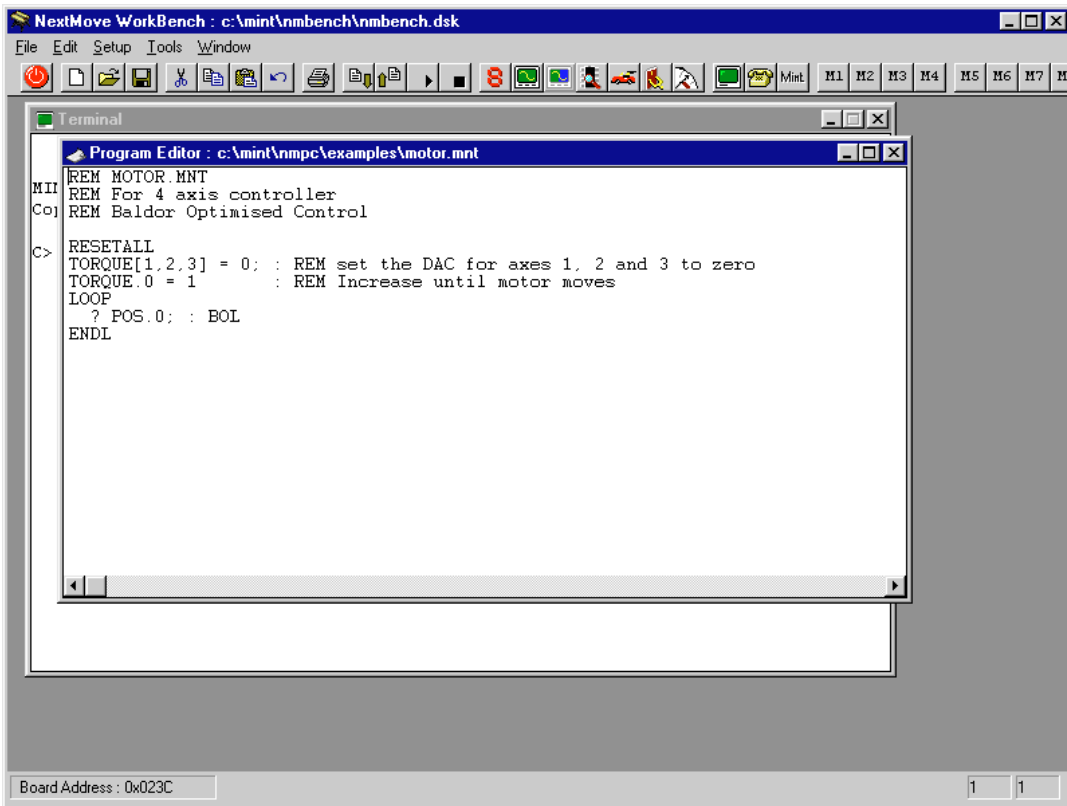



Figure 20: NextMove WorkBench program editor

- Press the *Download* button, . This copies the program to the controller.
- From the *Tool* menu, open a *Terminal Window*
- Type **RUN** and press the *return* key to execute the program.

The program will set the axes DAC output for axes 1,2 and 3 to zero. Increasing the TORQUE value on axis zero will increase the output of the DAC.

Starting with a value of 1, increase the torque value (**TORQUE**) until the motor starts to move. The motor should move in a positive direction i.e. the encoder position should increase. This can be seen in the *Motion Watch Window*. The torque value can be increased by editing the program in the *Program Editor* window and then re-downloading the program to the controller again. Alternatively the new **TORQUE** value can be entered at the command line from the *Terminal* window.

For Example:

```
TORQUE.0 = 5
```


will set the **TORQUE** value to be 5 (5% of the maximum DAC output).

If the position decreases the **demand/command** cables should be swapped over. Ideally, the **A** and **\bar{A}** signals on the encoder (and compliment signals) should be swapped, having first removed power from the amplifier.

Repeat the operation with a negative value of torque, for example:

```
TORQUE.0 = -1
```

The position should now decrease.

Repeat for the other axes. This will require two small modifications to the above program. To edit the program return to the *Program Editor* window and make the changes as follows:

- Line 2: Turn off the axes not currently being used. e.g. for axis 1
`TORQUE[0,2,3] = 0;`
- Line 3: Change the axis specified in the **TORQUE** command e.g. for axis 2
`TORQUE.2 = 1`

If the motor does not move with a torque value of 100 (100% demand output) check the following:

- The amplifier is enabled.
- The motor is connected.
- The amplifier is correctly configured.
- The controller is correctly connected to the amplifier
- There is a +10V output from the **demand+/command+** (for axis 0 there should be a voltage across J5, pins 13 and 14).

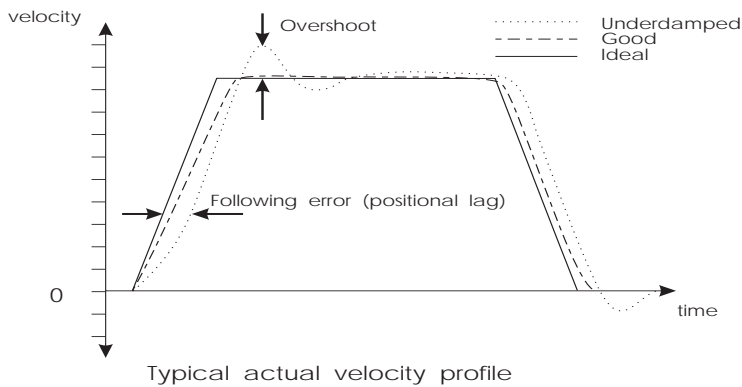
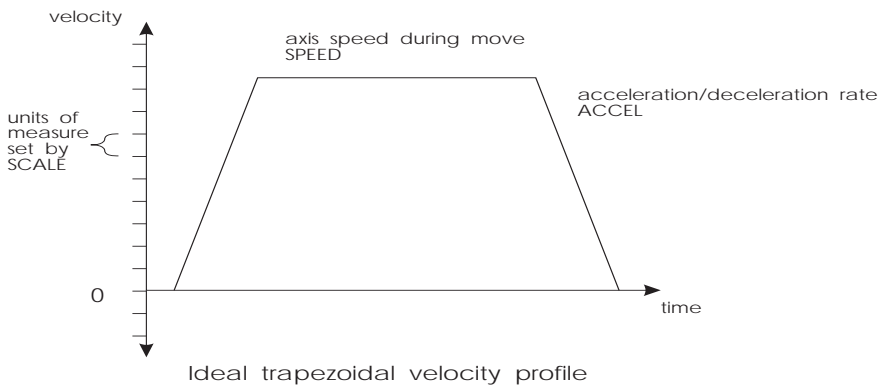
More suggestions are given in the fault finding guide at the back of this manual.

6.4.10 Setting System Gains

At the lowest level of control software, instantaneous axis position demands produced by the controller software must be translated into motor demands. This is achieved by closed loop control of the motor. The motor is controlled to minimise the error between demand and actual position measured with an incremental Encoder. Every 1ms (or optionally 500us using the **LOOPTIME** keyword) the controller compares desired and actual positions and calculates the correct demand for the motor. The corrective signal is calculated by a PIDVF (*Proportional, Integral, Derivative, Velocity Feedback and Velocity Feed forward*) algorithm.

Control could be achieved by applying a signal proportional to the error alone, but this is a rather simplistic approach. Imagine that there is a small error between demanded and actual position. A proportional controller will simply multiply the error by some constant, the *Proportional* gain, and apply the resultant to the motor via an amplifier. If the gain is too high this may cause overshoot, which will result in the motor vibrating back and forth around the desired position. As the gain is increased, the controller will present more resistance to positional error, but oscillations will increase in magnitude until the system becomes unstable.

To reduce the onset of instability a damping term is incorporated in the servo loop algorithm, called *Velocity feedback* gain. *Velocity feedback* acts to resist rapid movement of the motor and hence allows the proportional gain to be set higher before vibration sets in. (In some applications, the *velocity feedback* is handled by the amplifier, called a velocity servo). The effect of too high *proportional gain*, or too low *velocity feedback* gain is illustrated by the "Under damped" line in the figure below:



In *NextMove PC*, an alternative damping method is provided in the form of the *Derivative* of the **error** signal. *Derivative* action has the same effect as *velocity feedback* if the *velocity feedback* and *feedforward* terms are equal. In torque controlled systems, *Derivative* action is generally the preferred term.

With *Proportional* and *Derivative* action it is possible for a motor at rest at a set point to exhibit a small positional error (called *following error*). The controller multiplies the error by the proportional term to produce an applied corrective torque (in current control), but for very small errors the torque may not be large enough to overcome static friction. This error can be overcome by incorporating an *integral* term in the loop calculations. *Integral action* involves summing the error over time, so that motor torque is gradually increased until the *positional error* falls to zero. The speed at which *integral* action works is controlled by the *Integral gain*. *Integral* action is useful to eliminate steady state positional errors, but will result in reduced dynamic response for the system. For this reason, a software selectable option is provided so that the user can select that the Integrator is turned off during periods of constant velocity using **KINTMODE**.

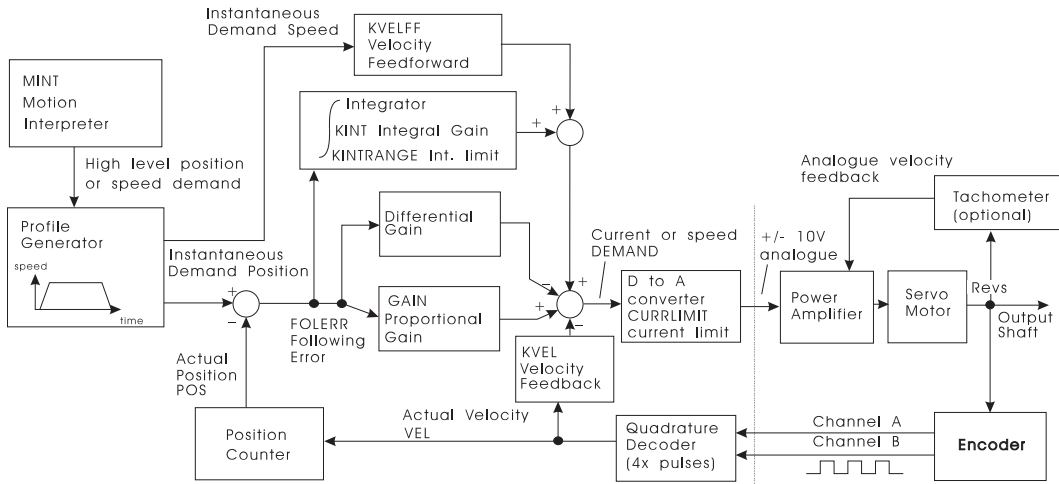
The final term in the control loop is *Velocity feed forward*. This is useful for increasing the response and reducing the following error, especially with velocity controlled servos.

Two types of servo amplifiers may be used with the controller:

- Current or torque amplifiers use the demand signal to control the current flowing in the motor armature and hence the torque of the motor.
- Velocity controlled amplifiers (velocity servo) use the demand signal as a servo speed reference.

For general purpose applications, the torque amplifier is cheaper and simpler to set up, but the velocity servo gives better control, especially in high performance applications. For torque amplifiers, velocity feedback must be used to stabilise the system, but this is not normally required for a velocity servo since it incorporates its own internal velocity feedback.

A block diagram of the complete control loop, showing controller, amplifier, motor and gearbox is presented below. The servo amplifier may be a simple current amplifier, or incorporate internal velocity feedback via a tachometer:



nm_3/1sg

Figure 21: Servo loop block diagram

The controller has five terms incorporating proportional, derivative, velocity feedback/feed forward and integral gains.

The equation of the loop closure algorithm is as follows:

$$\text{Demand} = \text{GN} \cdot e - \text{KD} \cdot \Delta e / \Delta \tau - \text{KV} \cdot v + \text{KF} \cdot V + \text{KI} \cdot \Sigma e$$

Terms	Description
GN	Proportional servo loop gain
KD	Derivative of error
KV	Velocity feedback gain
KF	Velocity feed forward gain
KI	Integral feedback
e	following error (quad counts)
v	actual axis velocity (quad counts/sample time)
τ	servo update period (sample time)
V	demand axis velocity (quad counts/sample time)

Tuning the closed loop involves selecting values for **GN**, **KD**, **KI**, **KV** and **KF** to provide the best performance for a particular motor/encoder combination and load inertia. In view of the diversity of applications, these values all default to zero and should be set up using the *System Gains* window within *NextMove WorkBench*. All gain values can be accessed via the Dual Port RAM while a move is in progress, this allows users to program *NextMove PC* to perform a trial move repetitively and then modify gain values during motion until an acceptable dynamic response is observed.

Two other functions, the *Integration limit* and *Current limit* are used to control the **demand/command** output. The *integration limit* determines the maximum value of the effect of integral action, $KI \cdot \Sigma e$. This is specified as a percentage (%) of the full scale demand output in the range of $\pm 10V$. Therefore if the integration limit = 25, the maximum effect of integral action is $\pm 2.5V$.

The *current limit*, so called for its use with current amplifiers, determines the maximum value of the demand output as a percentage of the full scale demand. Therefore if current limit = 50, the maximum demand output will be $\pm 5V$.

The encoder gain (measured in pulses/rev) is one factor that is hardware dependent but has a direct effect on the overall loop gain. Gains are controlled in MINT using the following keywords:

Keyword	Description
CURRLIMIT	Current limit setting
GAIN	Proportional gain
DGAIN	Derivative gain
KINT	Integral servo loop gain
KINTMODE	Integration mode during motion
KINTRANGE	Range limit for integrator % of max DAC output
KVEL	Velocity feedback gain
KVELFF	Velocity feedforward gain

6.4.11 Selecting Servo Loop Gains

All servo loop parameters default to zero, so the motor will have no power applied to it on power up. Most servo amplifiers can be set up in either current (torque) control mode or velocity control mode. The procedure for setting system gains differs slightly for each.

In order to check that the encoder and motor are wired up correctly, it is recommended that the motor is tested and commissioned ‘on the bench’ and not within the machine (the motor or gearbox shaft should be in free air).

To set up the closed loop system correctly, the following steps should be performed:

1. The integrity of the system wiring should have been checked as detailed in ‘Testing System Wiring’
2. Set-up system gains for satisfactory closed loop control. System gains may be set-up by an empirical method (the most common and practical approach) or by calculation followed by empirical fine tuning.

On line tuning of the system gains can also be accomplished by using the *Gains* option in the *NextMove PC WorkBench* and the *Software Oscilloscope* allows plotting of a response curve while a simple loop (such as a repeated small relative move) is running on the controller. Most servo amplifiers can be set up in either current (torque) control mode or velocity control mode. The procedure for setting system gains differs slightly for each.

6.4.12 System Gains for Current Control by Empirical Method

After having confirmed that the encoder and motor are correctly wired, start applying some *velocity feedback*, **KV**. Start with a value of 1 and increase it until you feel some resistance in the motor.

☞ $KV = 2$

For some motors, it may be necessary to apply fractional gains, all gain terms are floating point numbers.

Once the feedback gain has been set, apply some proportional gain, **GN**. Start off with a value which is a quarter of the feedback gain:

☞ $GN = KV / 4$

If the motor starts to vibrate, increase the *velocity feedback gain* (damping), **KV**, or decrease the proportional gain, **GN**. Increase proportional gain, **GN**, until the motor shaft becomes stiff.

Finally, set the velocity feed forward gain, **KF**, to the same value as the velocity feedback gain, **KV**.

An alternative to using **KV** and **KF** is to use the derivative term, **KD**. **KD** has exactly the same effect on the system dynamics as **KV** when $KV=KF$.

6.4.13 Mathematical Method Of Calculating System Gains

An alternative method of calculating system gains is described in the *Mathematical Method of Calculating System Gains* manual provided with the controller. This analysis provides a simple method of deriving starting values for current controlled systems without **KD**. It is provided for the interest of readers, but in practice can only provide a starting figure for gain values which can be determined more quickly by the empirical method.

6.4.14 System Gains for Velocity Control

Velocity controlled drives incorporate the velocity feedback term in the amplifier which provides system damping and therefore it is usually sufficient to have **KV = KD = 0** on the controller.

Usually, the value of the *proportional gain*, **GN**, will be less than with an equivalent current controlled system. Often a fractional value of *proportional gain* gives the best response. For example **GN=0.1** compared with **2** for current.

Correct setting of the *velocity feed forward gain*, **KF**, is important to get maximum response from the system. This is best performed using a storage oscilloscope (or the *NextMove WorkBench*) to record the tachometer output for fast point-to-point moves. This enables velocity/time profile for the motor to be seen in order to monitor actual acceleration and overshoot.

Referring to the servo loop block diagram, the velocity feed forward term is a block which takes the instantaneous speed demand from the profile generator and adds this to the output block. Because **KF** is a feed forward term, a very important difference between this and the other terms exists. **KF** is outside the closed loop and therefore does not have an effect on system stability. This means that the term can be increased to maximum without causing the motor to oscillate, provided that the other terms are set-up correctly.

In practice however, a very high value of **KF** is of no benefit to system performance. Instead, it should be set such that a demand of ω RPM from the profile generator results in a demand output to the velocity drive which gives ω RPM on the motor shaft (a 1:1 relationship).

When set-up correctly, **KF** will cause the motor to move at the demand speed from the profile generator. This is true without the PID terms in the closed loop doing anything except compensating for small errors in the position of the motor due to analog drift. This gives faster response to changes in demand speed, with lower *following errors*.

Example calculation of KVELFF:

In order to calculate the correct value for **KF**, you need to consider the workings of the servo loop closure algorithms. In the servo loop, speeds are expressed in quadrature counts/servo loop closure time. For instance, a speed of 100 is 100 counts every 1ms with *NextMove PC* (note that the default loop closure time is 1ms).

In this example the velocity of the servo is 3000RPM with a +10V input, and the encoder has 1000 counts per revolution.

At 3000RPM we require an analog voltage of +10V. This relates to:

$$\frac{3000}{60} = 50 \text{ revs per second}$$

Now the number of quadrature counts per loop closure time can be calculated by using the expression:

$$\frac{\text{speed_in_rev/sec} * \text{encoder_line_count} * 4}{\text{number_of_loop_closures_per_second}}$$

The factor of 4 is included since the controller counts every edge of the pulse train coming from the encoder **A** and **B** channels, which gives four times better resolution than the number of lines.

$$50 * 1000 * 4 / 1000 = 200 \text{ quadrature counts per servo loop closure time}$$

The DAC output has a resolution of 12 bits over the range -10V to +10V, therefore +10V = 2048 counts.

The feed forward term is therefore given by:

$$KVELFF = \frac{2048}{200} = 10.24$$

Increasing **KF** above the calculated value will cause the controller to have a *following error* ahead of the desired position. Decreasing **KF** below this value will cause the controller to have a more normal *following error* behind the desired position. The calculated value above should give zero *following error* in normal running.

It is possible to investigate the effect of *velocity feed forward gain* by jogging the motor at constant speed and printing out the *following error*, **FE**, for different values of **KF**. When attempting this, make sure that there is zero *integral gain* since this will cause the *following error* to tend to zero under steady state conditions, thereby negating the effect of changes in **KF**.

The software oscilloscope can be used in the same manner here as in section 6.4.13.

6.4.15 Fine Tuning System Gains

The above 'rules of thumb' for setting system gains, while adequate to get the system moving, will not provide the optimum response without further fine tuning of the system gains. The aim is to set the *Proportional Gain* as high as possible without getting overshoot or instability or hunting (the motor can be heard to buzz) on an encoder edge when stationary.

This is best achieved by attempting some short positional moves (say one revolution of the motor) with high accelerations and speeds and observing the response on an oscilloscope. The oscilloscope is normally connected to a tachometer on the motor, however with *NextMove PC* a software oscilloscope is available as part of the *NextMove PC WorkBench*.

Firstly an appropriate program should be running on the *NextMove PC* below is an example program:

```

AXES[0]      : REM commands refer to axes 0
SCALE = 4096 : REM Scale in revs assuming 1024 line encoder
SPEED = 10   : REM Max speed of 10 revs/sec
ACCEL = 100  : REM Accel of 100 revs/sec^2
DECEL = 100  : REM Decel of 100 revs/sec^2
LOOP
  MOVEA = 10 : Move +10 revs
  GO
  PAUSE IDLE
  WAIT = 250
  MOVEA = 0  : Move -10 revs
  GO
  PAUSE IDLE
  WAIT = 250
ENDL

```

This program moves axis zero to position 10 pauses for a quarter of a second and then moves back to position 0.

Using the *System Gains* window within the *NextMove WorkBench* the axis gains can be changed while the *NextMove PC* is running a program.

To use the software oscilloscope within the *NextMove WorkBench* the items to be plotted must first be added to the DPR watch window.

From the *Tools* menu select the *DPR WatchWindow*.

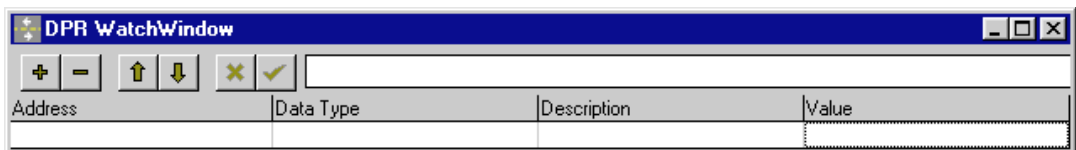



Figure 22: DPR WatchWindow

Clicking on the '+' button allows items to be added to the window. Any DPR location can be specified but for ease of use axis variables and general I/O locations are available on pull down menus.

Once the appropriate axis variables have been added to the *DPR WatchWindow* (for example; Measured Speed and Measured Position) they can then be plotted using the software oscilloscope.

Select *Software Oscilloscope* from the *Tools* menu or click the  icon. From the *Plots* tab select the elements from the *DPR WatchWindow* that are to be graphed. After clicking on OK the oscilloscope will start sampling when the *Start* button is clicked, and stop sampling and display the graph when the *Stop* button is clicked.

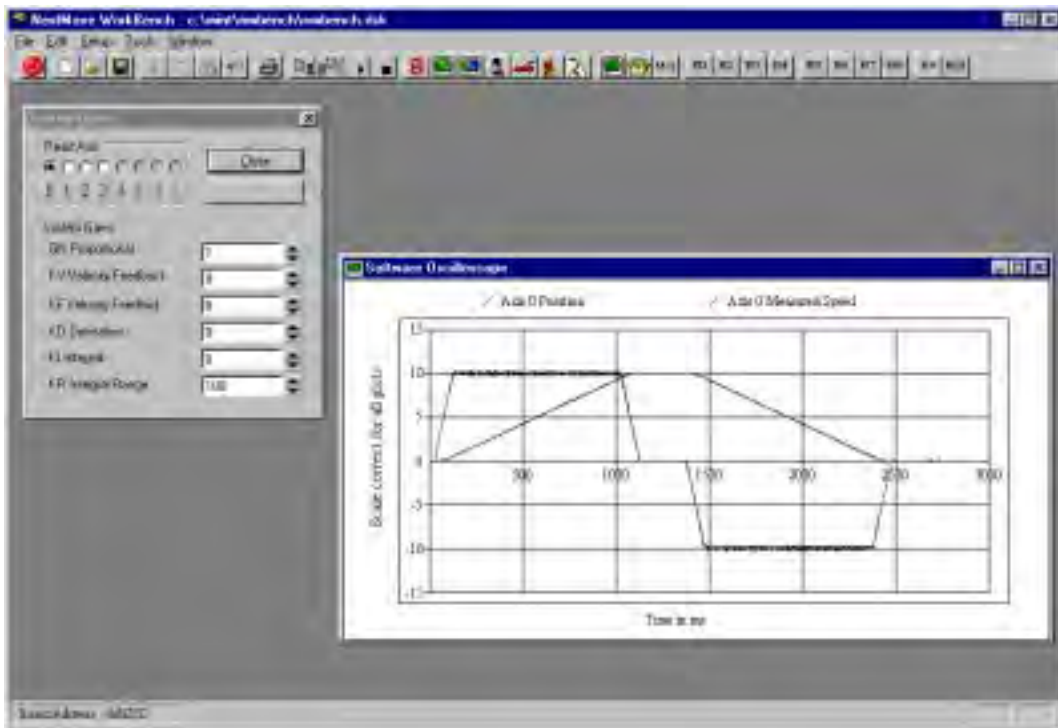


Figure 23: Fine tuning system gains

See section 11.13 for more details on the software oscilloscope.

In order to achieve an acceptable dynamic response, high gain values may be required, but these same values may cause the motor to hunt (a buzzing noise due to the motor vibrating on one encoder edge) when stationary. This can cause the machine to vibrate when at rest, especially if there is elasticity in the transmission such as a belt drive.

6.4.16 Eliminating Steady-State Errors

In systems where precise positioning accuracy is required, it is often necessary to position to within one encoder count. *Proportional gain*, **GN**, is not normally able to achieve this because a very small *following error* will only produce a small demand for the amplifier which may not be enough to overcome mechanical friction (this is particularly so for current controlled systems). This error can be overcome by applying some *integral gain*.

The integral gain, **KI**, works by accumulating *following error* over time to produce a demand sufficient to move the motor into the zero *following error* position. **KI** can therefore also overcome errors caused by gravitational effects, such as vertically moving linear tables, where with current controlled drives, a non-zero **demand/command** output is required to achieve zero *following error*.

Particular care is required when setting **KI** since a high value can cause instability during moves. The effect of **KI** should be limited by setting the maximum range of the integration (**KR**) to the minimum value which is sufficient to overcome friction or static loads. Typical values are:

KR = 5
KI = 0.1

where **KR** limits the integral term to 5% of the full DAC output range. **KI** is usually a factor of 10 less than *proportional gain*, **GN**. With *NextMove PC*, it is possible to set the Integrator such that it has zero effect on the system during periods of constant velocity (using the **KINTMODE** keyword) and therefore does not affect system dynamics.

6.5 Setting Up A Stepper Drive

The procedure for setting up a minimum stepper drive system is very much simpler than that for the servo system. The open loop control means there are no gains to set and there are no encoders to worry about (see Figure 17).

6.5.1 Limit And Stop Switches

For the purposes setting up the controller, these inputs can be ignored. This is because the software is initially set-up to ignore all errors, (except a following error). For information on configuring these inputs, please refer to section 6.6.

6.5.2 Pulse Generator Outputs

The controller has three outputs for each of four possible stepper motors - **PULSE**, **DIR** and **BOOST**. These are referred to *system ground (gnd)*. The **BOOST** output is a general purpose control line and is not used on all amplifiers. These outputs are found on the breakout board connector J6.

Before proceeding further with the set-up procedure, please ensure the amplifiers have been correctly set-up for the motor in accordance with the instructions in the amplifier manual.

For a one axis system connect axis 4 (remembering that the stepper axes are numbered from 4 to 7) to **PULSE.0**, **DIR.0**, **BOOST.0** and **gnd** (pins 37, 39, 41 and 43 respectively on J6 of the breakout board).

6.5.3 Amplifier Enable

The details of this connection do not change for a stepper motor system. It is only dependent upon the particular amplifier/drive set-up. Refer to section 6.4.4 in the servo drive set-up.

6.5.4 Ground Connections

A good ground connection to the controller is essential for noise immunity in industrial environments. Bad ground connection can be the cause of many problems, for instance loss of motor position.

Connect the ground connection stud on the amplifier to the ground connection stud on the break-out board using heavy duty cable. Then connect this stud to a common ground connection point using heavy duty cable. Please see the minimum system wiring diagram at the start of this system set-up section. See Figure 17.

6.5.5 Testing System Wiring

In order to check that the motor is wired up correctly, it is recommended that the motor is tested and commissioned 'on the bench' and not when installed in the machine.

The first step is to make sure the *WorkBench* is running and communicating with the *NextMove PC* card.

It is necessary for a terminal window to be open in order to issue commands to the controller.

- From the *Tools* menu select *Terminal*

To issue commands to the controller to move the motor in a positive direction:

1. At the prompt type:

```
jog.4=100
```

2. Press *Return*.

This moves the motor on axis 4 in a positive direction at reasonable speed. Looking in the *Motion WatchWindow* (from the *Tools* menu select *Motion WatchWindow* and ensure that the axis selected is **4**) it is possible see the position reading increase. (Note that this position reading is only a theoretical value as there is no feedback from the motor. It is possible to try and stop the motor shaft while it is turning and observe the position reading continuing to increase). Change the value to the right of the '=' sign to increase or decrease the speed of the motor.

Next change the speed to a negative value, e.g.

```
⊞ jog.4=-100
```

and press *Return*.

The motors should now spin in the opposite direction and the position reading decrease.

If the motor does not move, check the following:

- The amplifier is enabled.
- The motor is connected.
- The amplifier is correctly configured.
- The controller is correctly connected to the amplifier.
- That the correct speed/acceleration for your particular motor and drive set-up is being used.

More suggestions are given in section 12.

6.6 Input/Output

The *NextMove PC* controller brings out all Input/Output via a 100-pin high density shielded D-type connector on the board. The DIN rail mounted break-out board is provided for easy connection to your machine. Details of the NextMove PC Breakout Board can be found in section 13.

Summary of I/O

Reference	Number of connections	Description
dig_in_n	24	Digital inputs.
dig_out_n	12	Digital outputs.
Relay	3	Voltage free relay output for drive enable.
Ext_int	1	High speed external interrupt for position latching.
Encoder_n	4x6	Incremental Encoder Inputs phase A,B,Z.
dac_n	4	Analog outputs for servo drives.
Pulse/dir/boost_n	4x3	Stepper motor step/direction and boost outputs.
Can	2	Can bus twisted pair connections.
Adc_n	8	Analog inputs (8x12bit single-ended or 4x12bit differential).

The I/O configuration on NextMove PC is very flexible. The majority of I/O whether specific to motion control or for general purpose I/O is not defined by the hardware, but can be assigned to any user I/O pin via software. The attributes which can be attached to an I/O pin are defined below:

User Definable I/O Pins

Name	Description
stop Input	Motion Inhibit input - when asserted causes all axes to decelerate to a halt - used for machine guard schemes
error Input	Flags an external error to the controller - used to connect the drive healthy output to the controller
forward limit switch Axis n	Mechanical limit switch input in positive direction.
Reverse limit switch Axis n	Mechanical limit switch input in negative direction.
Home switch input Axis n	Home switch input.
Drive enable output Axis n	Output asserted in the event of an error on a particular axis - used to selectively enable and disable drives instead of the general enable relay.

These inputs may be configured to be active on a level or an edge. For level triggered inputs the active state can be set high or low. Edge triggered inputs can be configured to become active on positive, negative or both edges.

Eight analog input channels are also provided. These can be set up in software as eight single ended or four differential inputs. The range is also software selectable to be 0-5V or $\pm 2.5V$. In addition the range of each input can be extended by a factor of four (using jumpers) to $\pm 10V$.

To simplify the process of setting-up the controller defaults are assigned to each channel at the factory.

- All digital outputs are PNP.
- All digital inputs are level triggered, active high.
- Analog inputs are set to a range of 0-5V and are single-ended.

6.6.1 Changing the I/O Configuration

It is possible to change the default I/O configuration for *NextMove PC* using a number of MINT keywords.

MINT keyword	Comments
ACTIVEINLEVEL	Sets which inputs are active when high
ACTIVEOUTLEVEL	Sets which outputs are active when high
ADCMODE	Sets the mode in which an analog input channel is read
ANALOGUE	Reads the current analog value on a channel
DAC	Sets the value written to the DAC for an axis
DACMODE	Controls if the DAC is written in 14bit mode or 12bit emulation
DISDRIVE	De-activates the enable output for axis
ENABLE	Sets the state of the relay output
ENDRIVE	Activates the enable output for axis
ENOUTPUT	Defines the output to use for an axis drive enable output
ERRORIN	Defines the input to use for an external error input signal
FWLIMITIN	Defines the input to use for an axis forward limit switch input
HMINPUT	Defines the input to use for an axis home switch input
HOME	Returns the state of the home switch input for an axis
IN	Reads all inputs as a word
INACTIVEMODE	Sets which inputs are to be edge triggered
INx	Read the state of an input
LIMITF	Read the forward limit switch for an axis
LIMITR	Read the reverse limit switch for an axis
NEGEDGEIN	Sets which edge triggered inputs are to be active on a negative edge
OUT	Sets all outputs as a word
OUTx	Sets an output channel
POSEDGEIN	Sets which edge triggered inputs are to be active on a positive edge
REVLIMITIN	Defines the input to use for an axis reverse limit switch input
STOPINPUT	Defines the input to use for an axis stop switch input
STOPSW	Reads the state of the stop switch input for an axis

6.6.2 Example I/O Configuration

It is possible to change the default I/O configuration for *NextMove PC* using a number of keywords. Consider the following requirements for an example application:

- Five axes of control are required: 3 axes of servo and 2 axes of stepper.

- The servo axes have both **forward** and **reverse** limits. The stepper axes have only one **limit** input each.
- Each servo axis has its own **home** input. The stepper axes share the same input.
- The servo axes share the same **stop** and **external error** signals.
- The steppers axes share the same stop and **external error** signals.
- All servo axes have separate **drive enables**, stepper axes share the same.
- 5 user interrupts are to be installed which are rising edge triggered. Remaining inputs are level triggered.
- Outputs are active high.
- All axes are to decelerate at the **ERRORDECCEL** rate and not crash stop in the event of a **limit** error.

From the information above, the following I/O configuration is derived:

Input	Function	Trigger Mode
0	Reverse Limit Switch for axis 0	Level
1	Forward Limit Switch for axis 0	Level
2	Reverse Limit Switch for axis 1	Level
3	Forward Limit Switch for axis 1	Level
4	Reverse Limit Switch for axis 2	Level
5	Forward Limit Switch for axis 2	Level
6	Forward/Reverse Limit Switch on axis 4	Level
7	Forward/Reverse Limit Switch on axis 5	Level
8	Home Input for axis 0	Level
9	Home Input for axis 1	Level
10	Home Input for axis 2	Level
11	Home Input for axes 4,5	Level
12	Stop Input for axes 0,1,2	Level
13	Stop Input for axes 4,5	Level
14	External Error Input axes 0,1,2	Level
15	External Error Input axes 4,5	Level
16	Unused	Level
17	Unused	Level
18	Unused	Level
19	User Interrupt	Edge
20	User Interrupt	Edge
21	User Interrupt	Edge
22	User Interrupt	Edge
23	User Interrupt	Edge

Output	Function
0	Drive Enable for axis 0
1	Drive Enable for axis 1
2	Drive Enable for axis 2
3	Drive Enable for axis 4 and 5
4	Unused
5	Unused
6	Unused
7	Unused

All the dedicated inputs signals must be level triggered and not edge triggered. Given the above configuration, the following commands would be used to ensure the input signals are level triggered. Note by default all inputs are level triggered.

```
INACTIVEMODE = 0xF80000 : REM bits/inputs 19 - 23 edge triggered
POSEDGEIN = 0xF80000 : REM bits/inputs 19 - 23 RISING edge triggered
```

This will assign inputs **19** to **23** to be rising edge triggered. Inputs **0** to **18** will be level triggered. Note that the keywords accept a bit map. Hexadecimal notation has been used for clarity.

It is assumed that for the following examples, the macros *myservo* and *mystepper* have been defined:

```
DEFINE myservo = 0,1,2
DEFINE mystepper = 4,5
```

The following code segment will assign **forward** and **reverse** limits to the desired inputs. The command **HWLIMIT = 3** will decelerate the axes at the **ERRORDECEL** rate in the event of a limit being hit.

```
REVLIMITIN[myservo] = 0,2,4
FWLIMITIN[myservo] = 1,3,5
REVLIMITIN[mystepper] = 6,7
FWLIMITIN[mystepper] = 6,7
HWLIMIT = 3;
```

The following code segment will assign the **home** inputs to the desired inputs.

```
HMINPUT[myservo] = 8,9,10
HMINPUT[mystepper] = 11;
```

The following code segment will assign the **stop** inputs to the desired inputs. Note that the same input serves as a **stop** input to multiple axes.

```
STOPINPUT[myservo] = 12;
STOPINPUT[mystepper] = 13;
```

The following code segment will assign the error inputs to the desired inputs. Note that the same input serves as an **error** input to multiple axes.

```
ERRORIN[myservo] = 14;  
ERRORIN[mystepper] = 15;
```

The following code segment will assign the drive enables to the desired outputs. Note that the same output serves as a drive enable to both stepper drives.

```
ENOUTPUT[myservo] = 0,1,2  
ENOUTPUT[mystepper] = 3;
```

The above information has all been included in the file:

 **CONFIG.CFG**

File is in the MINT\NMPC\EXAMPLES directory.

