NextMove PC

Installation Manual

Issue 2.0

MN1257 11/98

Mint
THE MOTION LANGUAGE

Copyright Baldor Optimised Control Ltd © 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.
Telephone: +44 (0) 117 987 3100
Fax: +44 (0) 117 987 3101
email: sales@baldor.co.uk
Web site: www.baldor.co.uk

Baldor Electric Company
Telephone: +1 501 646 4711
Fax: +1 501 648 5792
email: sales@baldor.com
web site: www.baldor.com

Baldor ASR GmbH
Telehone: +49 (0) 89 90508-0
Fax: +49 (0) 89 90508-492

Baldor ASR AG
Telephone: +41 (0) 52 647 4700
Fax: +41 (0) 52 659 2394

Australian Baldor Pty Ltd
Telephone: +61 2 9674 5455
Fax: +61 2 9674 2495

Baldor Electric (F.E.) Pte Ltd
Telephone: +65 744 2572
Fax: +65 747 1708

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 Sections 2 and 7.

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	MN00277-000	First release of Installation Manual bringing together: • *NextMove PC* Overview • Getting Started Guide • *WorkBench* User Guide • *NextMove PC* Hardware Guide • Getting Started with CAN Peripherals
2.0	Sept 1998	MN00277-001	• Added KeypadNode 4 • Added NextMove PC Axis Expansion Board • General Tidy up
2.0	Nov '98	UM00508-000	• New reference raised

Read Me First 1

1.1 Key to Symbols Used in this Manual 2

Hardware Features 3

2.1 Technical Specification 5

2.1.1 Machine Control I/O 6

2.1.2 Miscellaneous and Mechanical Specification 7

Introduction to the NextMove Software 9

3.1 MINT Programming Language 10

3.2 MINT Programmer's Toolkit 17

3.3 NextMove WorkBench 18

3.4 Keyword Summary 20

Options 27

4.1 CAN Operator Panel (*Keypad*Node) 29

4.2 CAN I/O Nodes 30

4.3 The Encoder Splitter/Buffer Board 31

4.4 Axis Expansion Board 31

4.5 Breakout Board 31

4.6 MINT Interface Library 32

Getting Started 35

5.1 Installing the *NextMove PC* Controller 37

5.1.1 Placing the Card In the Computer 41

5.2 Installing The Software 43

5.2.1 Starting The *NextMove WorkBench* 43

5.2.2 Selecting A *NextMove PC* 44

5.2.3 Downloading an Application. 46

5.3 Minimum System Wiring Example 47
5.4 Setting Up A Servo Drive 49
5.4.1 Ground Connections 50
5.4.2 Encoder Connections 51
5.4.3 Amplifier Demand/Command Signals 52
5.4.4 Amplifier Enable 53
5.4.5 Limit And Stop Switches 54
5.4.6 Testing System Wiring 54
5.4.7 Checking The Encoder 54
5.4.8 Checking the Drive Enable 56
5.4.9 Checking Motor Polarity 56
5.4.10 Setting System Gain 58
5.4.11 Selecting Servo Loop Gain 62
5.4.12 System Gains for Current Control by Empirical Method 63
5.4.13 Mathematical Method Of Calculating System Gain 64
5.4.14 System Gains for Velocity Control 64
5.4.15 Fine Tuning System Gains 66
5.4.16 Eliminating Steady-State Errors 68
5.5 Setting Up A Stepper Drive 69
5.5.1 Limit And Stop Switches 69
5.5.2 Pulse Generator Outputs 70
5.5.3 Amplifier Enable 71
5.5.4 Ground Connections 71
5.5.5 Testing System Wiring 72
5.6 Input/Output 73
5.6.1 Changing the I/O Configuration 75
5.6.2 Example I/O Configuration 76

Introduction to the MINT™ Programming Language 79

6.1 The Configuration File 80
6.2 The First MINT Program 82
6.2.1 Program Narrative 85
6.3 A Simple Cut to Length Feeder 86
6.3.1 Configuration Buffer FEEDER.CFG 88
6.3.2 Program buffer FEEDER.MNT 89
6.3.3 Cut To Length Program Narrative 93
6.3.4 Using Batch Numbers 97
6.4 X-Y Teach and Replay Program 97
6.5 Software Gearbox Example - Coil Winding Machine 100
6.5.1 Program Narrative 102
6.5.2 Remote Operation Using the COMMS array 102
6.6 Infeed Packaging Machine 103

7. Hardware Guide 107

7.1 Operating Environment 109
7.2 Connection to the PCB 109
7.3 Digital I/O 111
7.3.1 Digital Inputs 112
7.3.2 Fast Interrupt 114
7.3.3 Digital Outputs 115
7.3.3.1 Output Polarity Selection 115
7.3.3.2 NPN Outputs with Higher Voltages 117
7.4 Analog I/O 117
7.4.1 Analog Inputs 117
7.4.1.1 Input Range Selection 118
7.4.1.2 Differential Mode 118

7.4.2 Analog Outputs (Drive Demand/Command) 119
7.5 Encoder Interface 120
7.6 Error Relay 121
7.7 Stepper Drive Outputs 121
7.8 Communications 123
7.8.1 Serial Port 123
7.8.2 CAN Bus 124
7.9 Reset State 126
7.9.1 Communications 126
7.9.2 Digital Outputs 127
7.9.3 Analog Outputs 127
7.9.4 Pulse Generator Outputs 127
7.10 LEDs 128
7.11 Expansion Board 129
7.12 Miscellaneous 129
7.12.1 Emulator Connection 129
7.12.2 System Watchdog 129
7.12.3 Interrupt Level 130

NextMove PC Breakout Board 133

8.1 Encoders: P1, P2, P3 and P4 135
8.2 CAN: JP7 and JP8 136
8.3 Terminal Block JP5 137
8.4 Terminal Block JP6 138

CE Marking 141

9.1 Directives 142
9.1.1 Machinery Directive 89/392/EEC 142
9.1.2 Low Voltage Directive 72/23/EEC 142

9.1.3 EMC Directive 89/336/EEC 142
9.2 EMC Performance of *NextMove PC* 143
9.3 Conditions of CE Marking 144

NextMove PC Axis Expansion Board 145

10.1 Connection Information 151
10.1.1 Servo Signals 151
10.1.2 Master Encoder 152
10.1.3 Stepper Outputs 153
10.2 Servo Breakout Board 154
10.2.1 J3: Demand/Command Signals 154
10.2.2 Encoders: P2, P3, P4 and P5 155

Getting Started with CAN Peripherals 157

11.1 Network Possibilities 159
11.2 Quick Start 159
11.2.1 Jumper setings 160
11.2.2 Connections and Configuration 161
11.3 NextMove PC and CAN Peripherals 163
11.3.1 Selection of CAN Channel 163
11.3.2 Selection of CAN Baud Rate 163
11.3.3 Selection Of Node ID 163
11.3.4 Network Termination 165
11.3.5 Static Configuration 166
11.3.6 Normal Operation 169
11.4 An Example Network 169
11.5 Using A *KeypadNode* 171
11.6 MINT Keyword Summary 173

MINT Programmer's Toolkit..... 175

12.1 Starting The WorkBench..... 176
12.2 The Main Toolbar..... 177
12.3 Selecting the Board Address..... 178
12.4 Downloading an Application..... 180
12.5 Terminal..... 180
12.6 Program/Configuration Editors..... 182
12.7 Squash..... 183
12.8 Macros..... 185
12.9 Setting up Gains..... 186
12.10 Motion WatchWindow..... 187
12.11 The Comms Window..... 187
12.12 Dual Port Ram WatchWindow..... 188
12.13 Software Oscilloscope..... 189
12.14 Axis WatchWindow..... 191
12.15 Digital I/O..... 192
12.16 Analog Inputs..... 193
12.17 Project Files..... 194
12.18 DOS Utilities..... 195

Trouble Shooting Guide..... 197

Bibliography..... 201

Read Me First

1

This chapter provides an overview of the manual and symbols used throughout this document.

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 2 through to 4. This contains an overview of the *NextMove PC* controller, MINT™, the motion programming language and options available for *NextMove PC*.
- **Getting Started**: Sections 5. This section contains details of getting started with *NextMove PC*, checking wiring and tuning the motors.
- **Introduction to MINT Programming Language**: Section 6. This provides an overview of the MINT motion programming language.
- **Hardware Guide**: Sections 6 through to 10 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 11. This provides a description of configuring *NextMove PC* with a number of CAN Peripherals.
- **WorkBench**: Section 12. Provides information on *WorkBench* for *NextMove PC*.

It is advisable to read Section 5, Getting Started, if unfamiliar with *NextMove PC*.

1.1 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.

Hardware Features

2

This chapter provides a technical overview of NextMove PC, the 8 axis PC based motion controller.

- ◊ Introduction to NextMove PC.
- ◊ Technical overview
- ◊ Dimensions.

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.

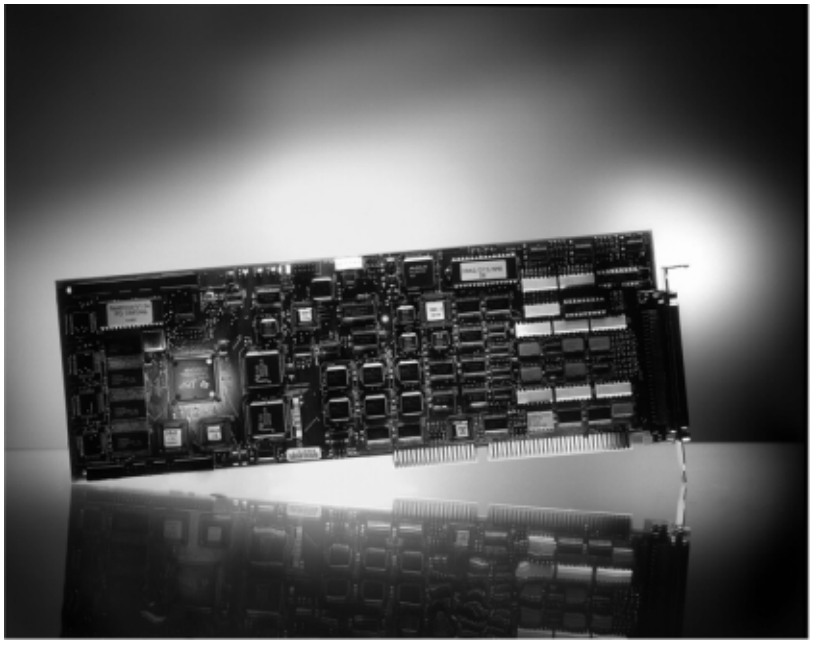


Figure 1: *NextMove PC* Controller

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 with 12 bit analog outputs and incremental encoder feedback; 500 or 1000 microsecond servo loop update rate.
- Four axes of stepper motor control 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 30µs.
- 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, ±2.5V or ±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.

2.1 Technical Specification

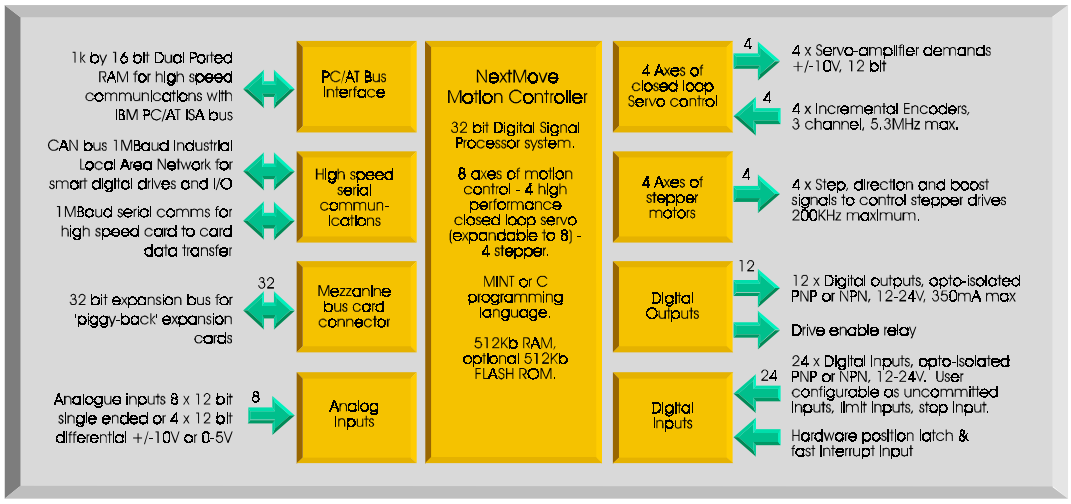


Figure 2: Technical Overview

2.1.1 Machine Control I/O

User Digital Inputs:	• 24 input lines. • PNP or NPN opto-isolated (factory set). • 12 to 24V ±20% on, 3V max off. • Software programmable as **limit**, **home**, **stop** or **error** inputs for any axis.
User Digital Outputs:	• 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**/**error** output for any axis. • 50mA continuous source on all channels. 750mA total per bank
Fast Interrupt:	• 1 input per controller. • 5V CMOS, capacitivly decoupled. • Latches position of all axes within 30 microseconds on falling edge.
Enable (Relay) Output:	• 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.
Analog Inputs:	• 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.
Analog Outputs:	• 1 output per axis for motor **demand/command** signal (software configurable). • ±10V output (±0.1%). • 12 bit resolution (4.88mV per bit).

Encoder Inputs:	• 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:	• Step, direction and boost. • Open collector outputs. 100mA @ 24V. • 200kHz maximum output frequency.

2.1.2 Miscellaneous and Mechanical Specification

Power Input:	• +5V at 850mA • ±12V at 250mA • User Power: +12V to 24V at 850mA
Weight:	• Approximately 305g / 0.67lb
Operating Temperature:	• 0 - 40°C / 32 - 104°F Ambient
Warranty:	• 5 year return to manufacturer.

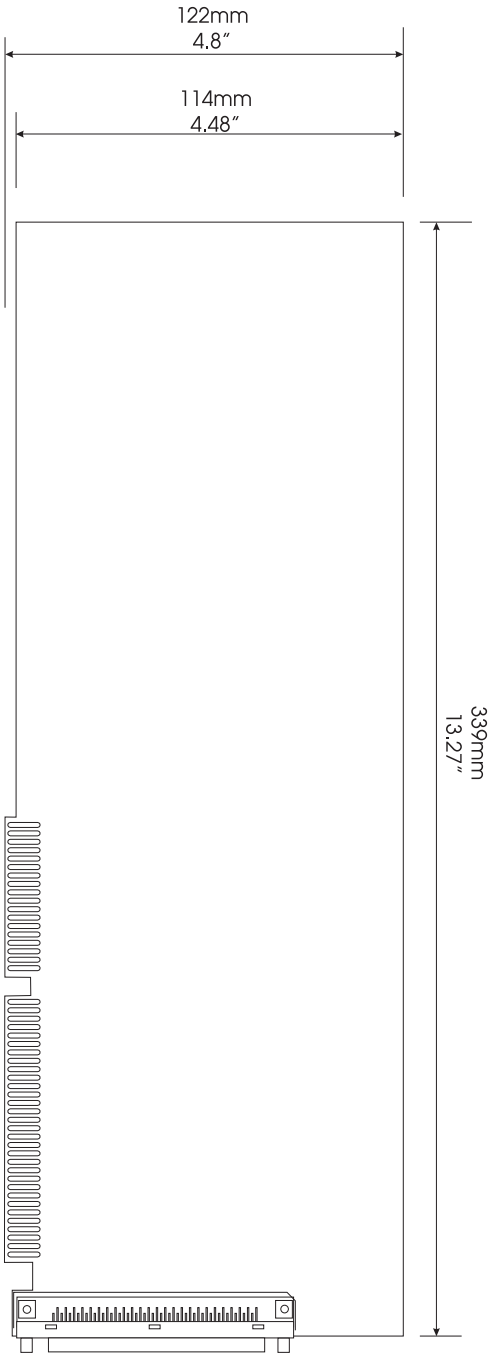


Figure 3: *NextMove PC* Overall Dimensions

Introduction to the NextMove Software

3

This chapter provides an introduction to the programming of NextMove PC and the available tools for Windows.

- ◊ Introduction to the MINT, the Basic programming language covering language basics such as the configuration and program files.
- ◊ Introduction to the NextMove WorkBench, the Windows front end for NextMove PC.
- ◊ Summary of all MINT keywords.

Motion applications can be realized using a number of different programming methods:

1. MINT: Basic programming language
2. Embedded 'C': Using an appropriate compiler from Texas Instruments, a 'C' application can be written to linked to the MINT Motion Library. This is then downloaded to NextMove for execution.
3. Host application: Using the MINT Interface Library (a Windows 16 or 32 bit Dynamic Link Library (DLL)), motion programs can be written in a number of Windows development environments, such as Microsoft Visual Basic, Microsoft Visual C++ or Borland Delphi.

Programming option 1 is supported as standard with the product. Options 2 and 3 are available as part of the MINT Interface Library (see section 4.6). A cut down version of option 3 is available as standard on the enclosed CD.

The following sections give an over of the MINT, the Basic language and the NextMove PC WorkBench, a 16 bit Windows front end for NextMove.

If developing 32 bit applications in Windows NT, please contact your local distributor for details on the MINT Interface Library.

3.1 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 12.7.

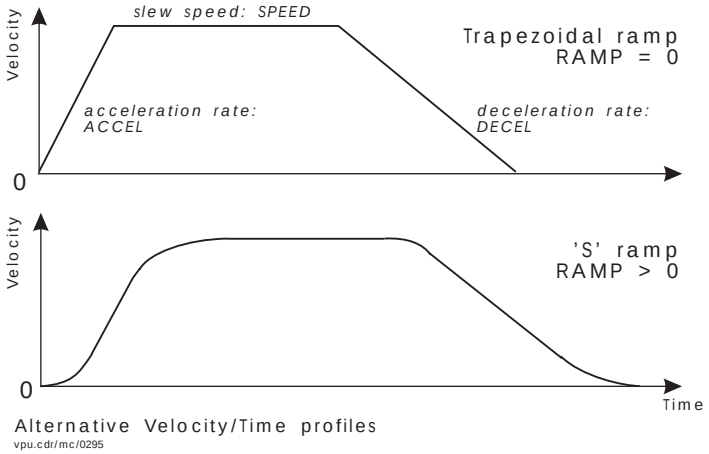


Figure 4: Trapezoidal move with S ramping applied

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 Configuation 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 30µs. 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.

```
#IN0
  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.

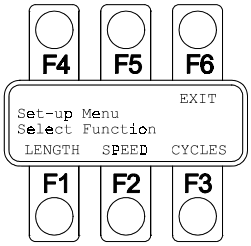


Figure 5: Set-up Menu

The MINT for NextMove Programmer's Manual contains full information on the MINT programming language.

3.2 MINT Programmer's Toolkit

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

- MINT application file for the controller. This is the executable file which provides the real time motion control and MINT execution for NextMove.
- *NextMove Workbench* which provides a Windows front end with terminal, editor and software oscilloscope. See section 12.
- DOS Utilities such as NMVER for establishing the firmware version number. See section 12.18.
- MINT Interface Library Standard Edition allowing a Windows front end application to be written which communicates with the MINT application over Dual Port RAM.
- Various example programs

There are also some uninstalled files on the CD.

- Various user manuals in .PDF format. These can be later editions to the printed documents :
  - ◊ MINT Interface Library Standard and Developer Release
  - ◊ MINT Programmers Manual

- ◊ NextMove PC WorkBench User's Guide
- ◊ MINT for NextMove Revision History
- ◊ cTERM and MINT Utilities for Windows and DOS
- ◊ CAN Peripherals Installation Manual and Programming Manual

- A PDF reader (Acrobat)
- Disk images for the MINT Programmer's Toolkit should a CD-ROM player not be available on the final machine housing NextMove PC.

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

3.3 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.

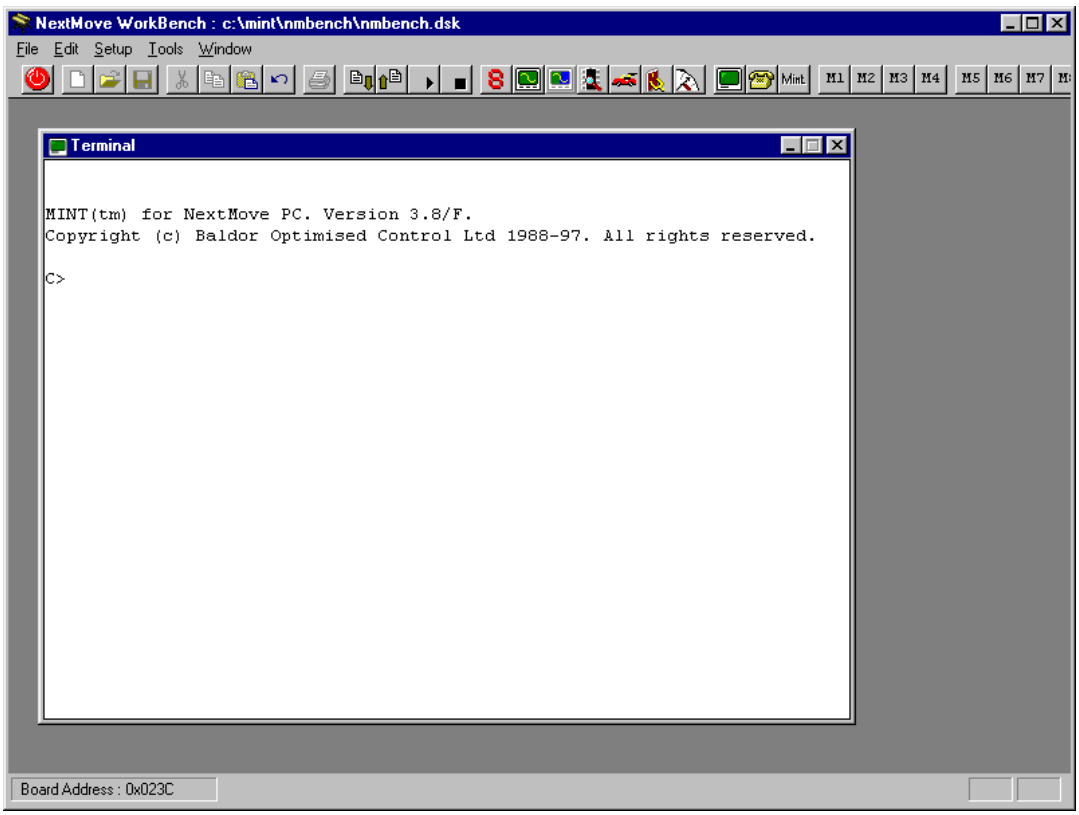


Figure 6: *NextMove WorkBench* Terminal Window

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.

3.4 Keyword Summary

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

Keyword	Description
Program Constructs	
:	statement separator
REM	comment
Subroutines	
#	subroutine label
#CAN	can event/error subroutine
#FASTPOS	fast position interrupt routine
#IN0..#IN23	interrupt subroutines on digital inputs
#ONERROR	on error subroutine
#STOP	stop subroutine on stop input
#TIMER	timer event subroutine
Conditional Statements	
IF DO {ELSE} ENDIF	block IF structure
IF THEN	standard IF THEN statement
PAUSE	pause program execution until condition is true
Loops	
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
Program Flow	
AUTO	automatic execution of program
GOSUB	branch to subroutine
GOTO	continue execution from label
END	end program execution
RETURN	return execution from subroutine
Terminal Input/Output	
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

Keyword	Description
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

Motion Keywords

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

Keyword	Description
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
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
VECTORR/VR	vector move relative
VEL/VL	return instantaneous axis velocity
ZERO/ZR	set current axis position as zero

System and Configuration Keywords

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

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

Keyword	Description
REMOTERESET/RR	reset a CAN node
REMOTESTATUS/RS	read /clear remote status
STATUSNODE/SN	read node number in error
Comms Functions	
COMMS	read/write value using protected communications mode
COMMSMODE/CMM	enable RS232/RS485 multi-drop comms
Data Capture	
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
Line Editor Commands:	
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
Relational and Mathematical Operators	
+ - / * < > <= >= <> =	
~	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

Options

4

NextMove PC supports a number of options, including an axis expansion card and CAN I/. This chapter summarises those options.

- ◊ CAN Operator Panel.
- ◊ CAN I/O modules.
- ◊ Axis expansion board.
- ◊ MINT Interface Library for embedded ‘C’ development and Windows NT.

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 – 3 Axes	KPD002-502	CAN bus based 27 key keypad and 4 line LCD display
CAN Operator Panel – 4 Axes	KPD002-505	CAN bus based 41 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 connecter via screw down terminals.
Breakout Cable 1.0 m	CBL005-501	1.0 m 100 way cable to atttach *NextMove PC* to Breakout boad.
Breakout Cable 1.5 m	CBL005-502	1.5 m 100 way cable to atttach *NextMove PC* to Breakout boad
Breakout Cable 3.0 m	CBL005-503	3.0 m 100 way cable to atttach *NextMove PC* to Breakout boad
Miscellaneous:		
NextMove PC manual	MN1257	Includes software
MINT Interface Library	SW1259	Libraries for Windows developers

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

4.1 CAN Operator Panel (*Keypad*Node)

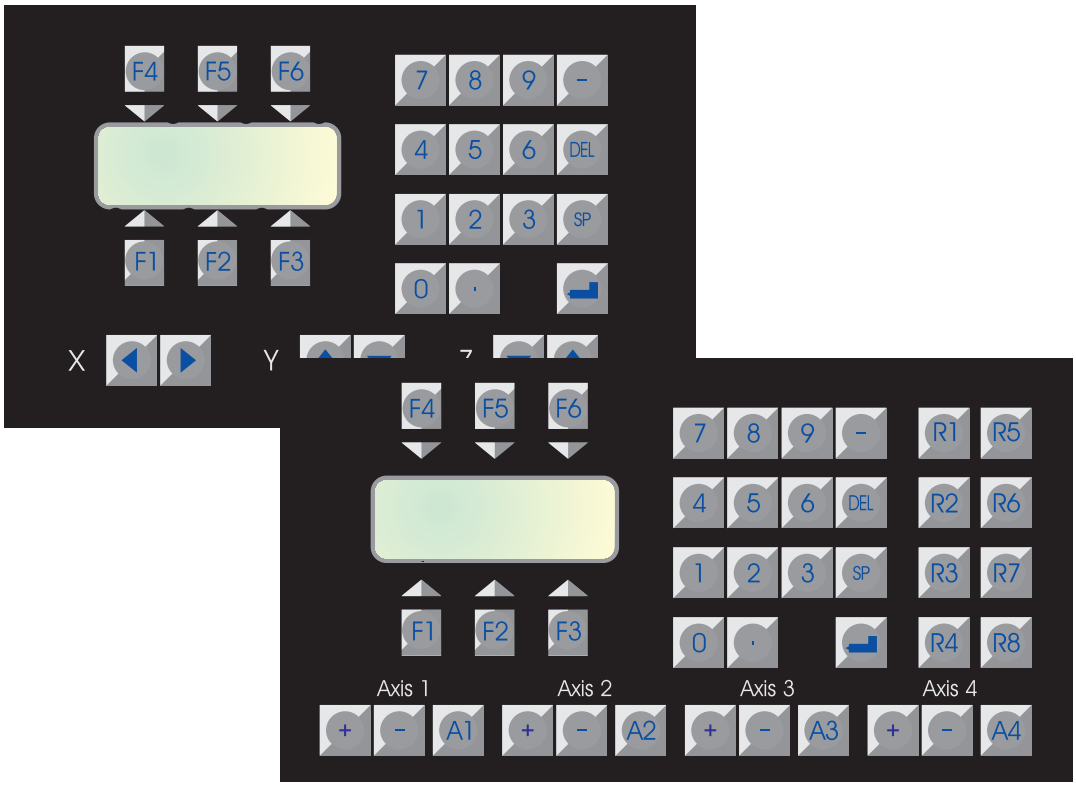


Figure 7: KeypadNode (3 and 4 axis units)

The CAN Operator Panel (*Keypad*Node) provides a general purpose operator panel suitable for stand-alone machines of all types. *Keypad*Node 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.

*Keypad*Node 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.

4.2 CAN I/O Nodes

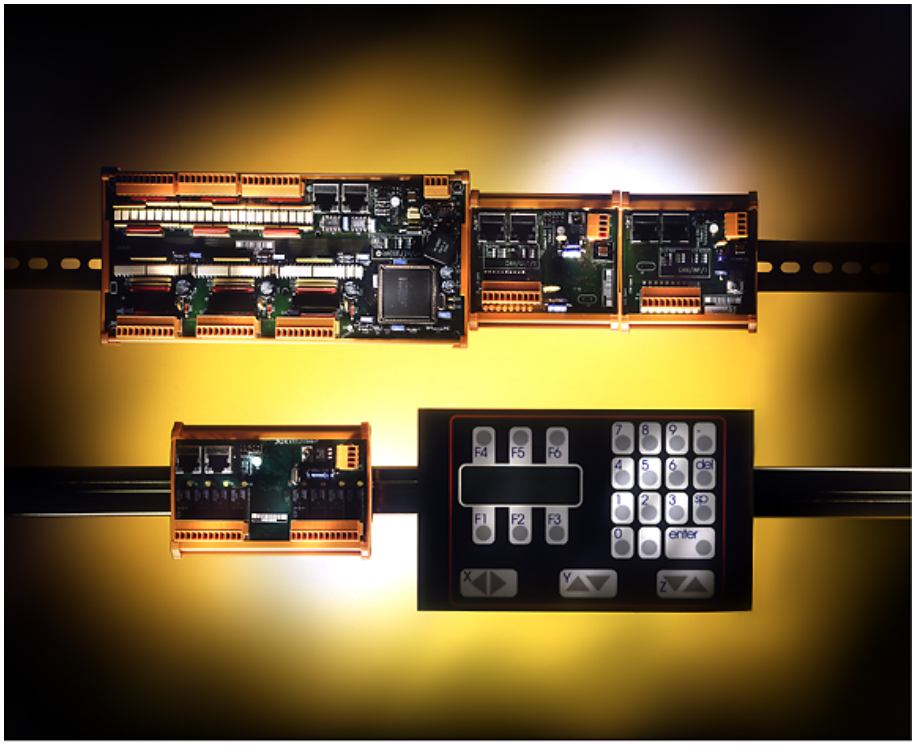


Figure 8: *io*NODE Product Family

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 *Keypad*Node, the CAN bus operator panel. Refer to Section 10 for further information on CAN Peripherals.

4.3 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 'daisychaining' an encoder signal from a master across a number of controllers.

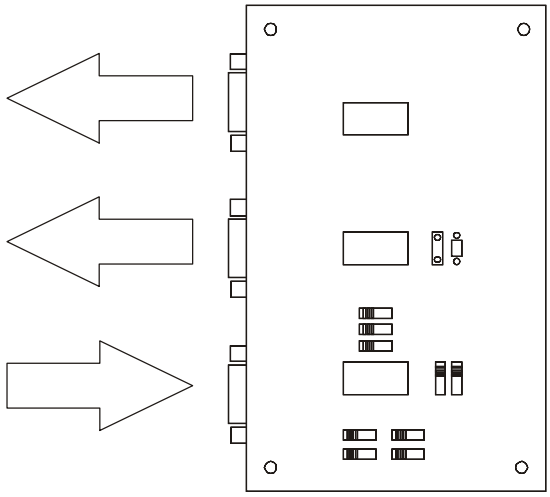


Figure 9: Encoder Splitter/Buffer Board

4.4 Axis Expansion Board

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 10.

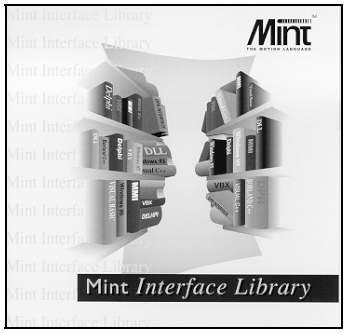
4.5 Breakout Board

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 8.

4.6 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* .
- Embedded ‘C’ applications using the Texas Instruments C31 ‘C’ compiler. ‘C’ programs can be written and linked to the motion core of NextMove and downloaded to NextMove for execution.
- Windows NT is supported in full using a device driver specially developed for NextMove PC. A 32 bit version of the NextMove WorkBench, known as MINT WorkBench is also supported.

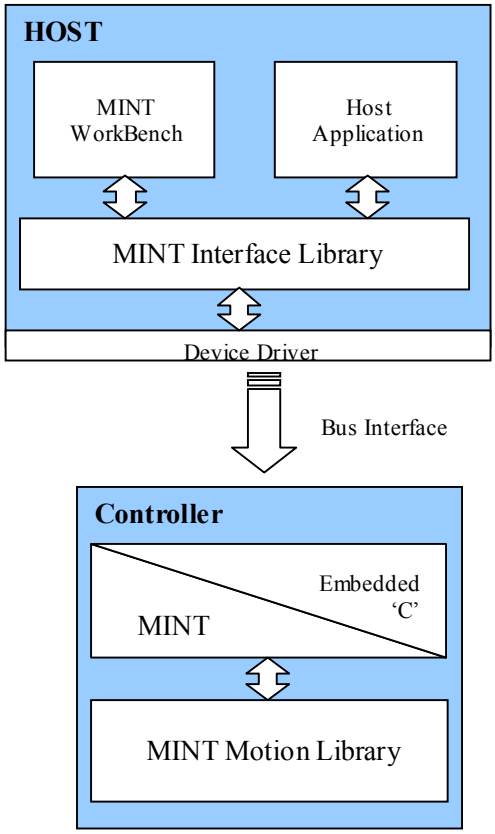


Figure 10: Simplified View of the NextMove Software Architecture

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

Programming Examples:

MINT Example:

```
MOVEA.2 = 100
GO.2
v = POS.1
```

Embedded 'C' Example:

```
setMOVEA( 2, 100 );
GO( 1,2 );
getPOS( 1, &v );
```

Host C++ Example:

```
CNextMovePC myNMPCI ( 0,0x200 );
myNMPCI.setMOVEA( 2, 100 );
myNMPCI.GO( 1,2 );
myNMPCI.getPOS( 1, &v );
```

Visual Basic Example:

```
Dim hNM&
HNM =
  createNextMovePCHandle(0,0x200)
setMOVEA hNM, 2, 100.0
GO1 hNM, 2
getPOS hNM, 1, v
```

Contact a local sales office for further details on the MINT Interface Library.

Getting Started

5

This chapter is step by step guide to setting up a NextMove PC servo and stepper control system. Basic familiarity with PC's and the Windows environment is assumed.

- ◊ Installing NextMove PC and accompanying software into the PC.
- ◊ Connecting NextMove PC to a servo or stepper system.
- ◊ Introduction to servo systems and tuning. Determining system gains and fine tuning.
- ◊ Connecting limits and home inputs.

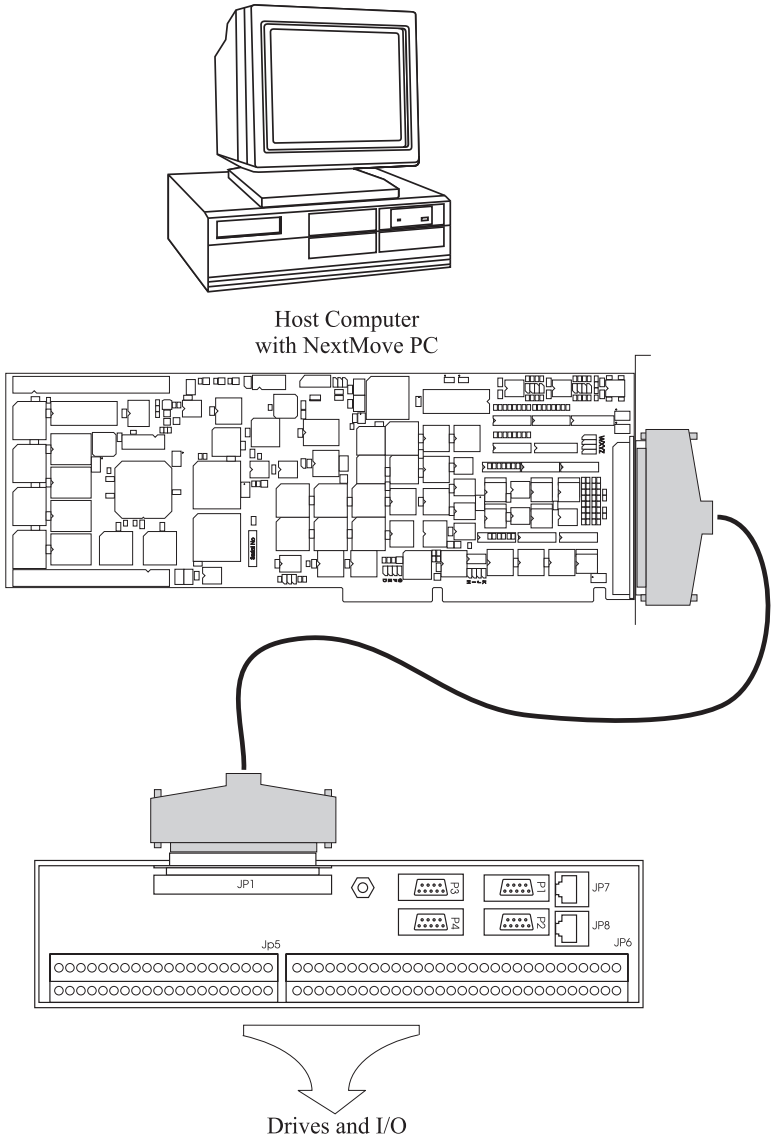


Figure 11: NextMove PC Connected to Breakout board

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 6 of this manual and covered in more detail in the *MINT for NextMove Programmers Manual*. Refer to section 12 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.

5.1 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.

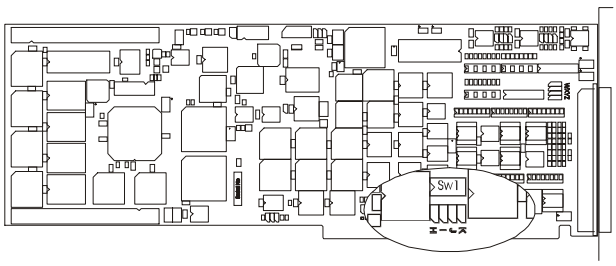
The host must have a spare **full length** card slot. It must also be an AT type machine as this card will not work correctly in a PC-XT. The card dimensions conform to the ISA standard except in respect of the 100 way D-type connector on the metal bracket, which hangs below the bottom edge of the card 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.*

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.

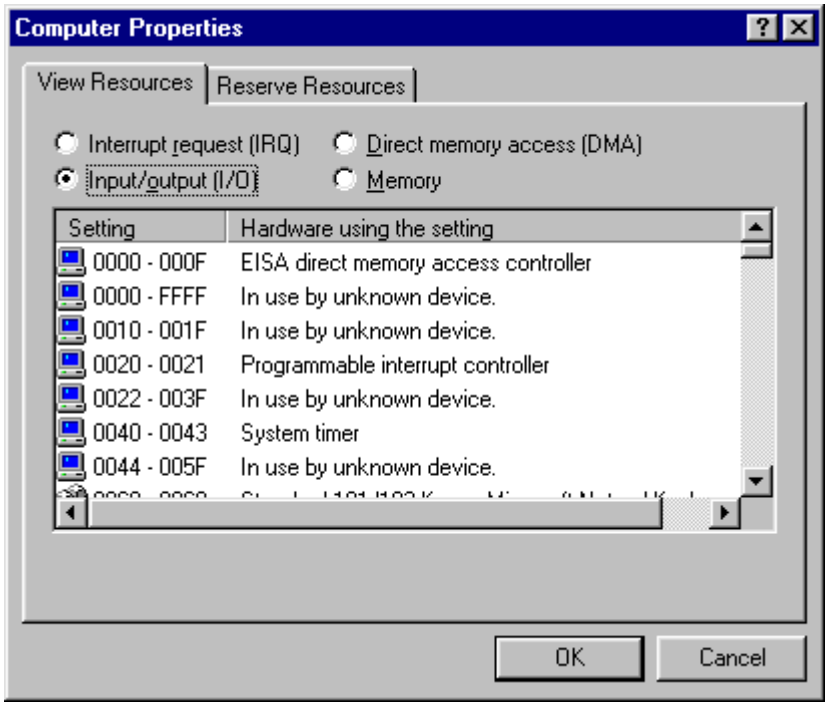
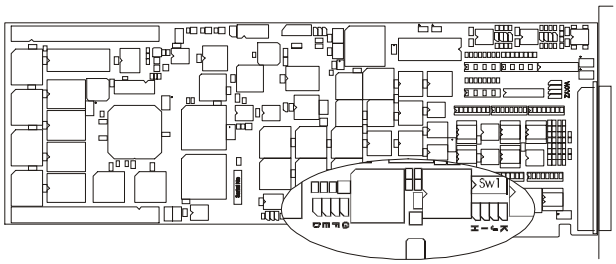


Figure 12: Windows '95 I/O map

Various interrupt levels are jumper selectable. The jumpers are located near the address selection DIP switches.



Jumper Fitted	PC Interrupt	Typical Use
I	IRQ3	Serial Port 2
J	IRQ5	Parallel Port 2
K	IRQ9	Spare
G	IRQ10	Spare
F	IRQ11	Spare
E	IRQ12	Spare
D	IRQ15	Spare

If no interrupts are required, then all interrupt jumpers should be removed.

5.1.1 Placing the Card In the Computer

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.

Locate an unused 16 bit, full length slot.

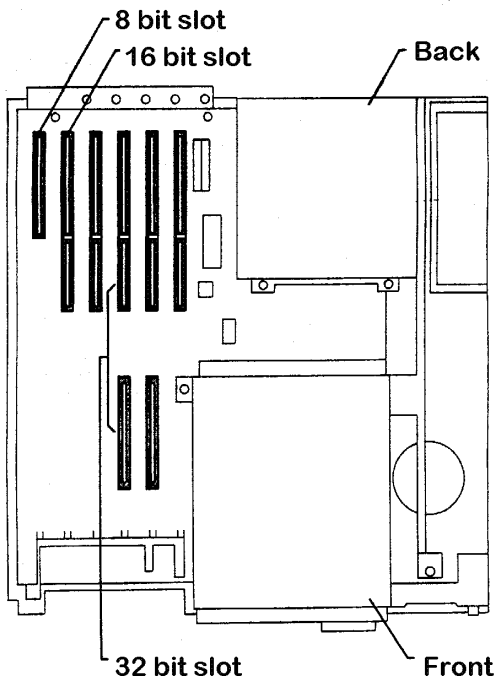


Figure 13: PC Slots

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.

1. Remove the cover from the slot, and save the screw for use later.
2. Discharge any static electricity from hands.
3. Remove the card from its protective wrapper. Do not to touch the gold contacts at the bottom of the card.
4. 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.
5. 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.
6. Anchor the card in place using the screw removed earlier.
7. Replace the computer cover and screws.
8. Reconnect any cables and power cords that were disconnected or unplugged.

5.2 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 *NextMove 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`**.

If NT is used as the operating system, contact the local distributor for details on the MINT Interface Library, which provides full support for Windows NT. This manual assumes that either Windows 3.x or Windows 95 is used.

5.2.1 Starting The *NextMove WorkBench*

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

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

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

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

When the *NextMove WorkBench* is run for the first time, a blank screen is presented as shown in Figure 14. The bottom left corner of the screen should display a Board Address of the controller. This will be 0, and it will be necessary to change this value to the value set in section 5.1. The following section describes the two basic steps necessary to communicate with the *NextMove PC* controller.

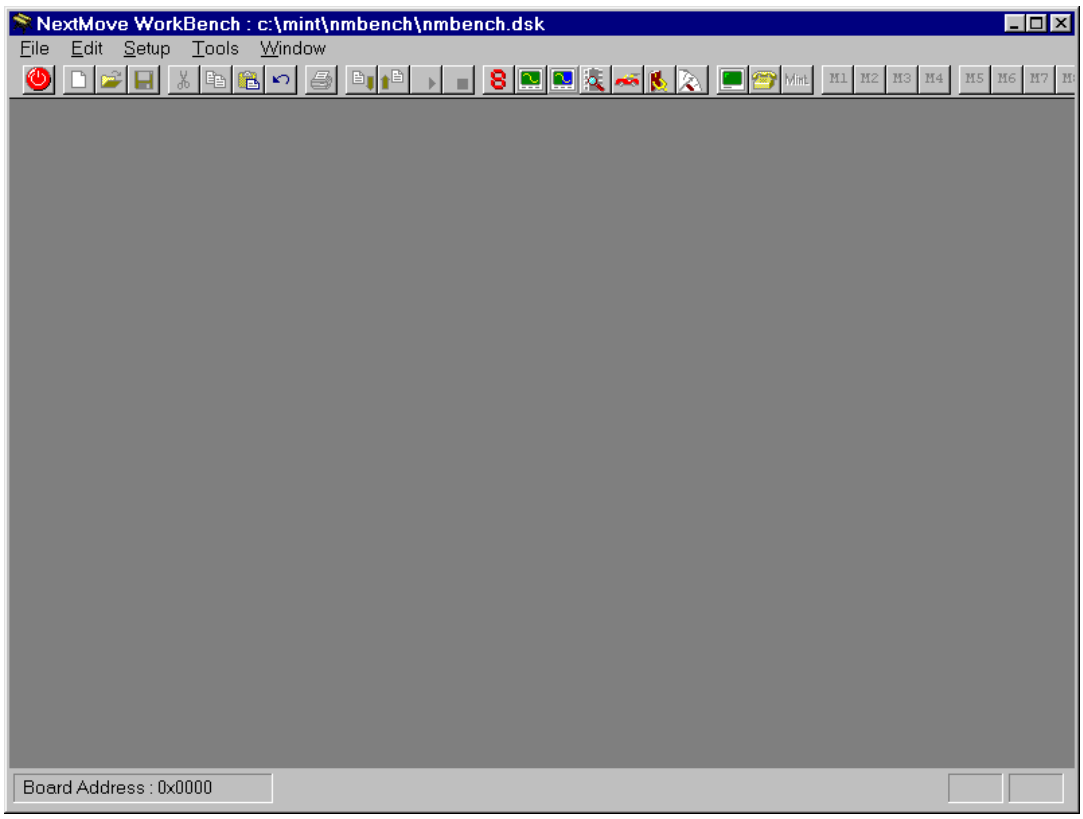


Figure 14: *NextMove WorkBench* Default Window

5.2.2 Selecting A *NextMove PC*

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

- Click on the *Board Address* label at the bottom of the *NextMove 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)*.

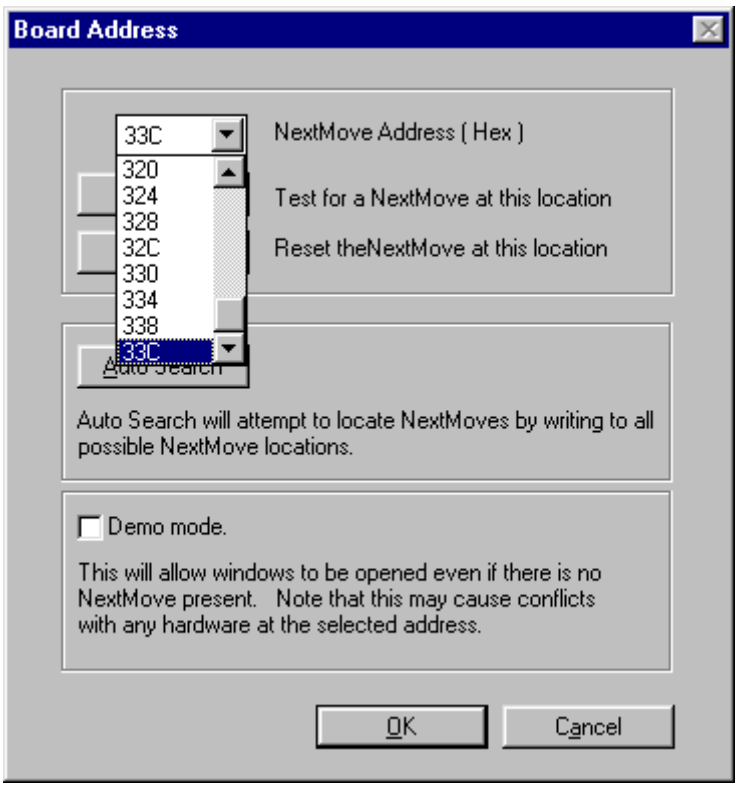


Figure 15: NextMove Address Window

Select the address to that set in section 5.1.

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*.

5.2.3 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.

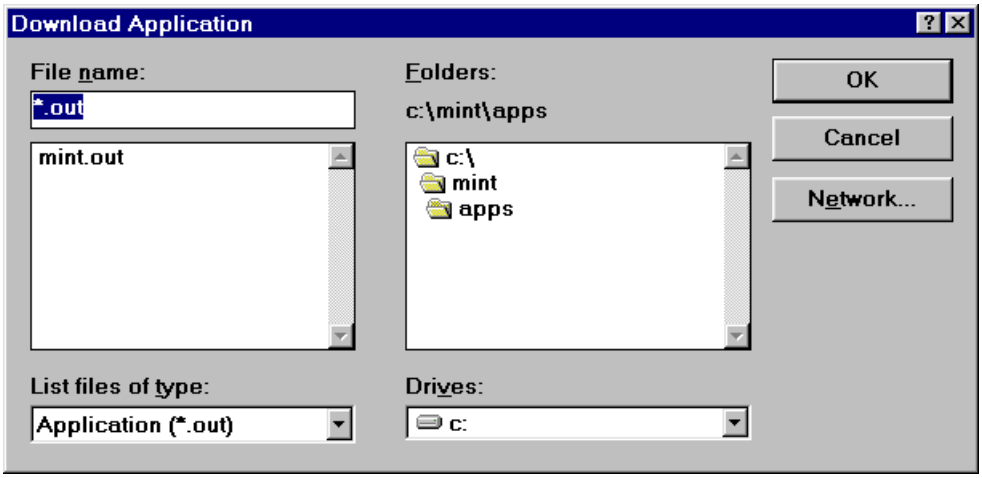


Figure 16: Download Application

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.

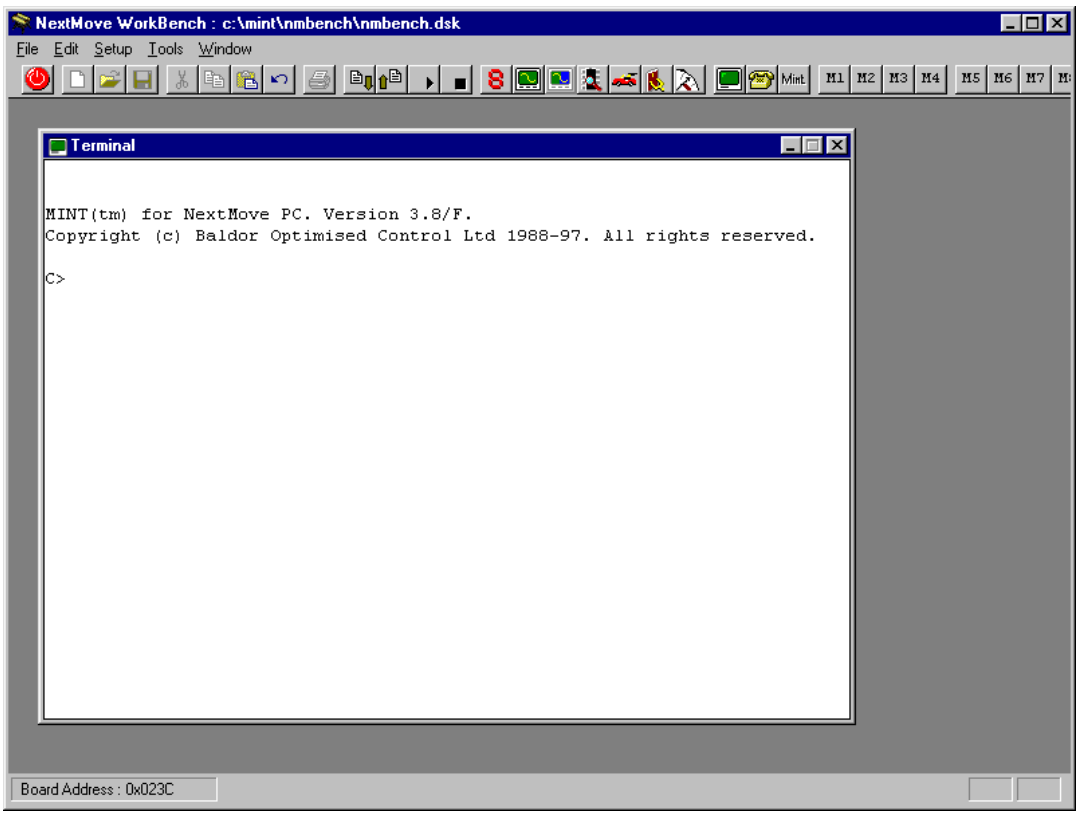


Figure 17: Terminal window

5.3 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 8.

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.

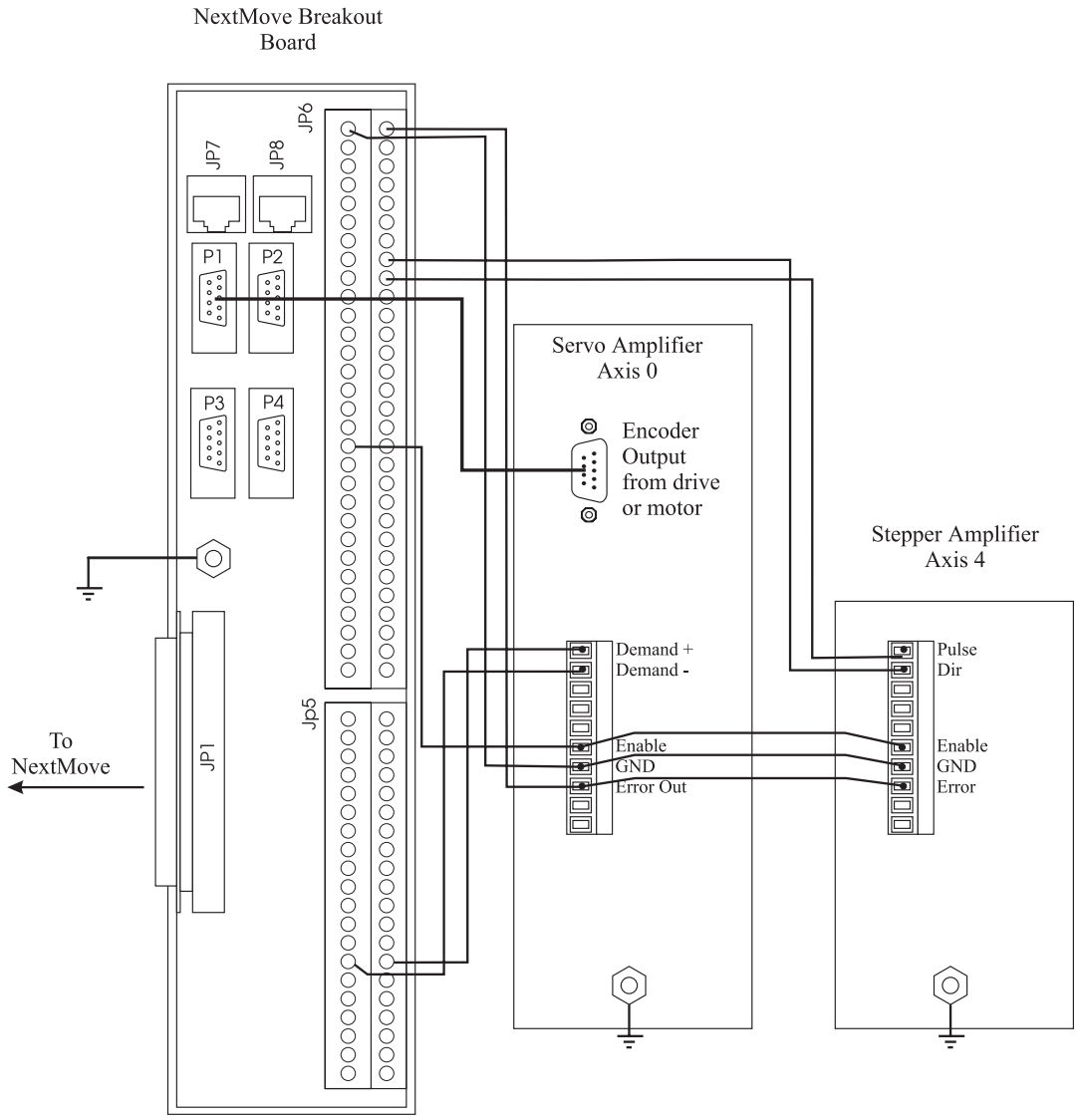


Figure 18: Minimum System

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 8.

5.4 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 19.

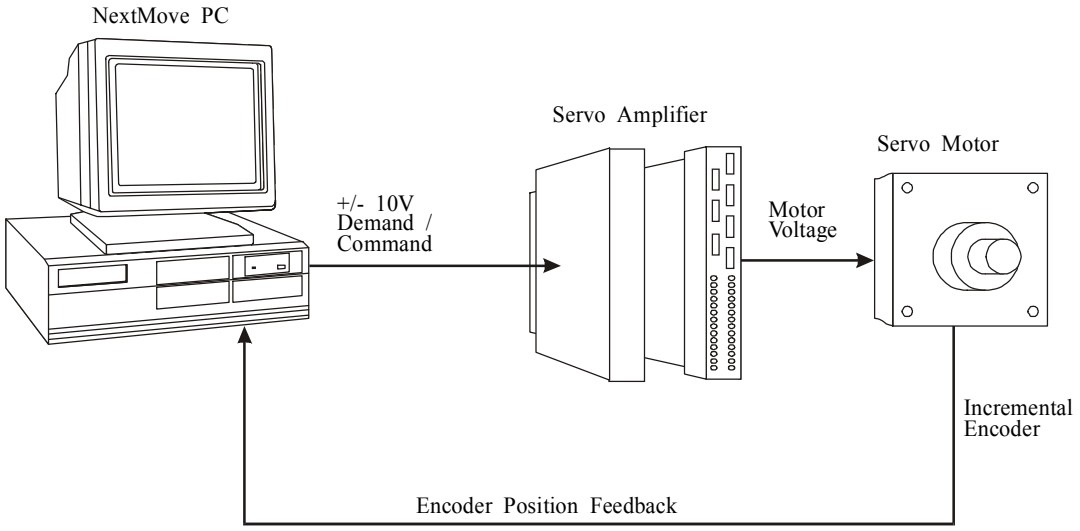


Figure 19: Typical closed loop positioning system

5.4.1 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 18.

5.4.2 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.

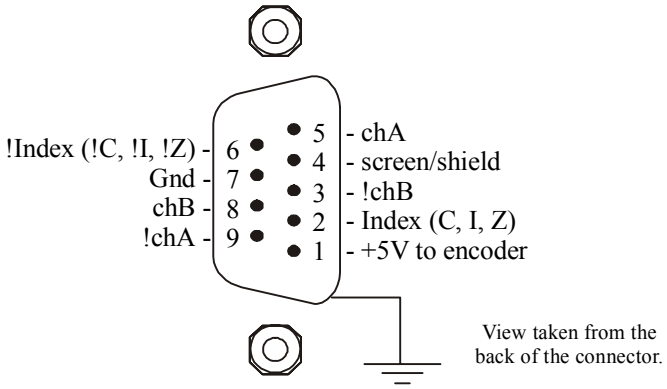


Figure 20: Encoder Pinout

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 7.5).

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 Figure 20. 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 8.1 for full details on the encoder circuit).

5.4.3 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 break-out board connector J5. The connections are shown in the table:

Pin	Signal	Function
13	**demand 0**	Demand Signal 0/Analog Output 0
14	**Agnd**	Demand Reference/ Analog Ground
15	**demand 1**	Demand Signal 1/ Analog Output 1
16	**Agnd**	Demand Reference/ Analog Ground
17	**demand 2**	Demand Signal 2/ Analog Output 2
18	**Agnd**	Demand Reference/ Analog Ground
19	**demand 3**	Demand Signal 3/ Analog Output 3
20	**Agnd**	Demand Reference/ Analog Ground

For the single axis system connect the axis 0 **demand+/command+** to **demand 0** (pin 13) and **demand-/command-** to **agnd** (pin 14).

5.4.4 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. The signals are found on JP6 on the breakout board.

Pin	Signal	Function
53	**relay-com**	Error Relay Common Connection
54	**relay-nc**	Error Relay Normally Closed Connection
55	**relay-no**	Error Relay Normally Open Connection

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.

5.4.5 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 5.6.

5.4.6 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.

5.4.7 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*.

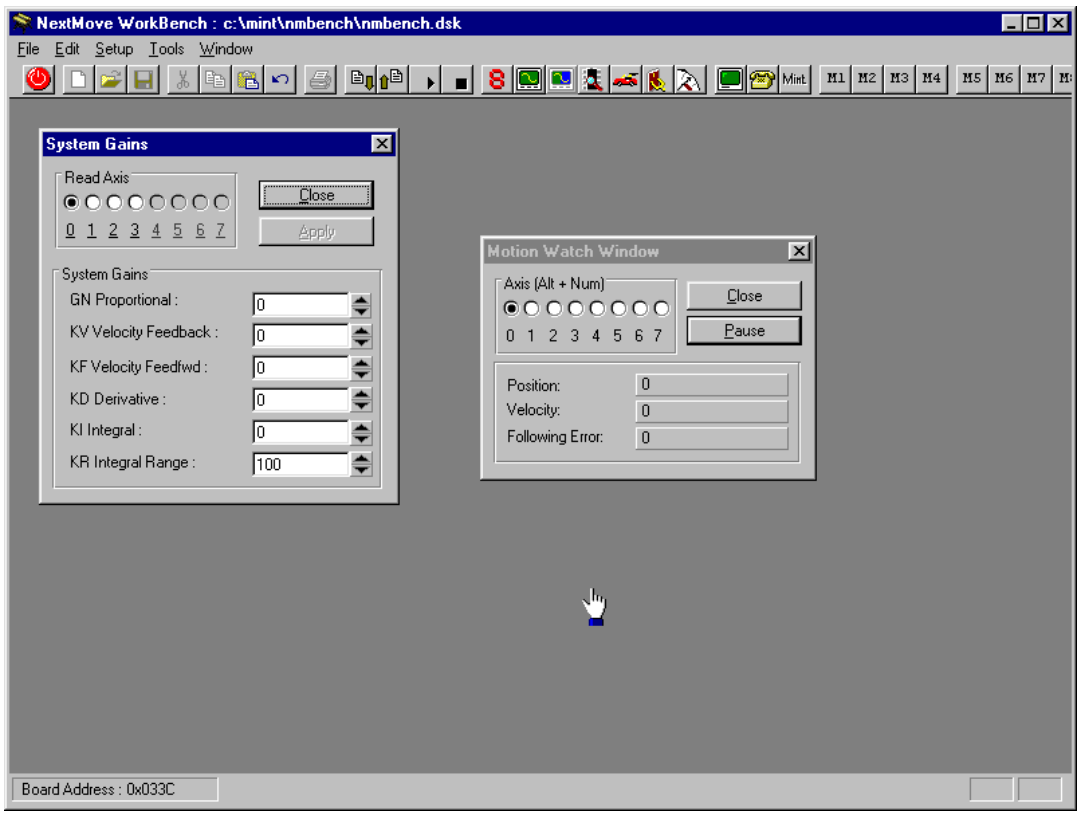


Figure 21: Gains

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.

5.4.8 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.

5.4.9 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.

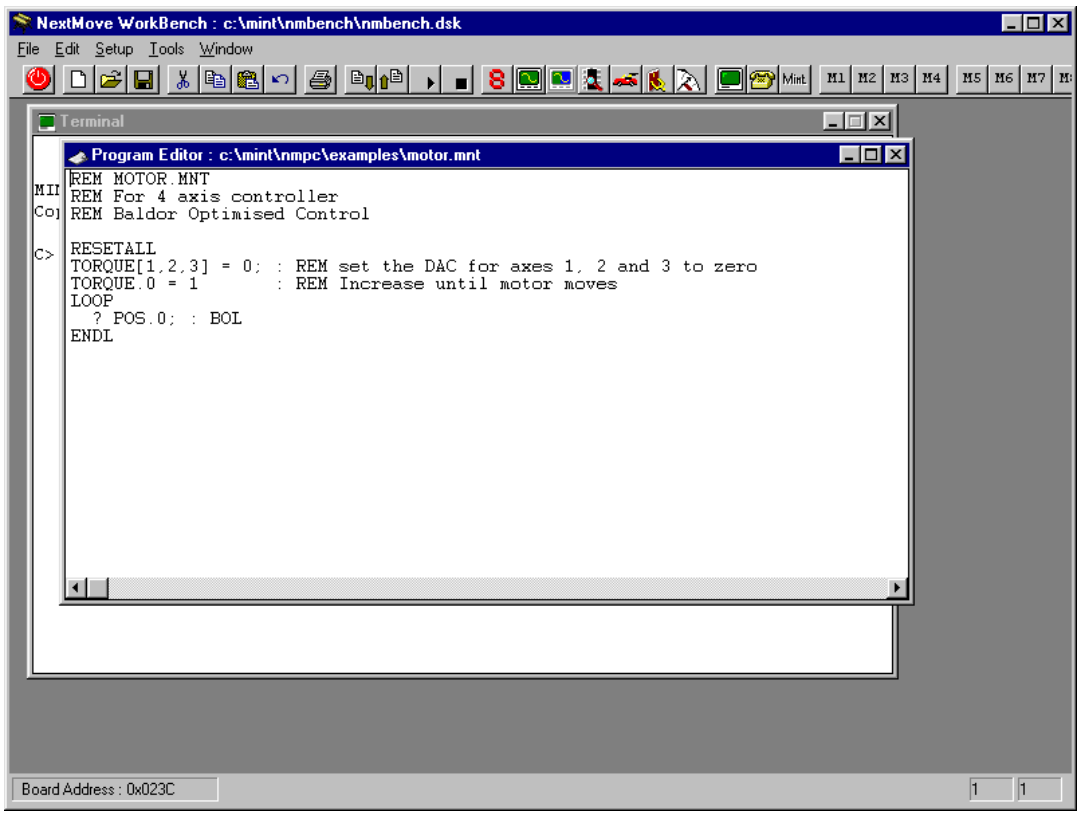


Figure 22: *NextMove WorkBench* program editor

- 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 **B** 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).

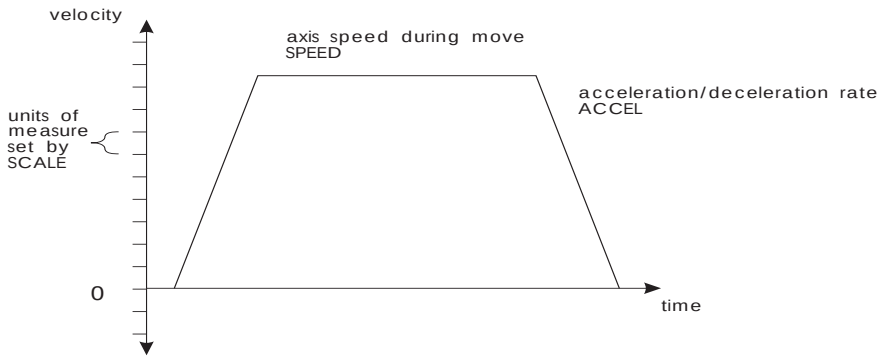
More suggestions are given in the fault finding guide at the back of this manual.

5.4.10 Setting System Gain

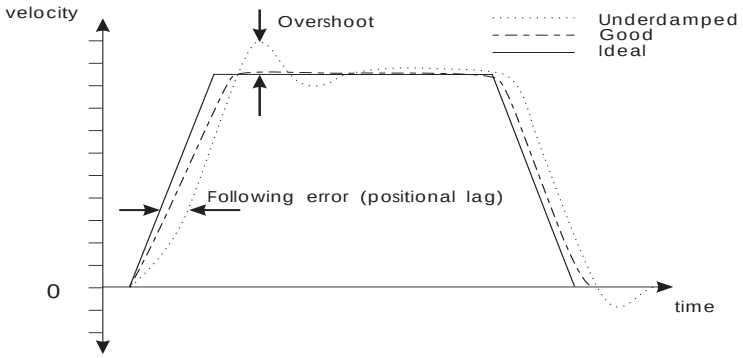
At the lowest level of control software, instantaneous axis position demands produced by the controller software must be translated into motor demands. This is achieved by closed loop control of the motor. The motor is controlled to minimise the error between demand and actual position measured with 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:



Ideal trapezoidal velocity profile



Typical actual velocity profile

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:

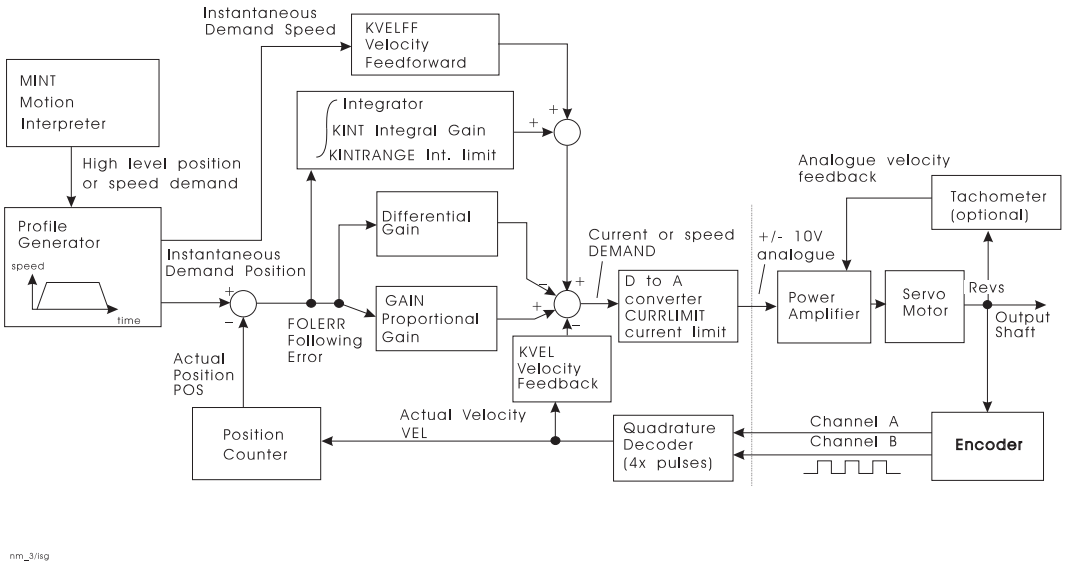


Figure 23: Servo loop block diagram

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:

$$\mathbf{Demand = -N.e - KD.\Delta e/\Delta\tau - KV.v + KF.V + KI.\Sigma e}$$

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.Σe. This is specified as a percentage (%) of the full scale demand output in the range of ±10V. Therefore if the integration limit = 25, the maximum effect of integral action is ±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 ±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

5.4.11 Selecting Servo Loop Gain

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.

5.4.12 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**.

5.4.13 Mathematical Method Of Calculating System Gain

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.

5.4.14 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.

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

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 5.4.13.

5.4.15 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 : REM Move +10 revs
  GO
  PAUSE IDLE
  WAIT = 250
  MOVEA = 0  : REM 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* W*atchWindow*.

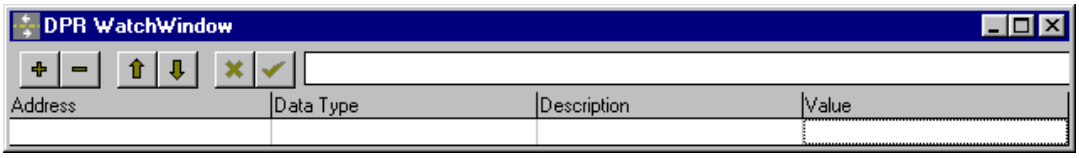


Figure 24: DPR WatchWindow

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.

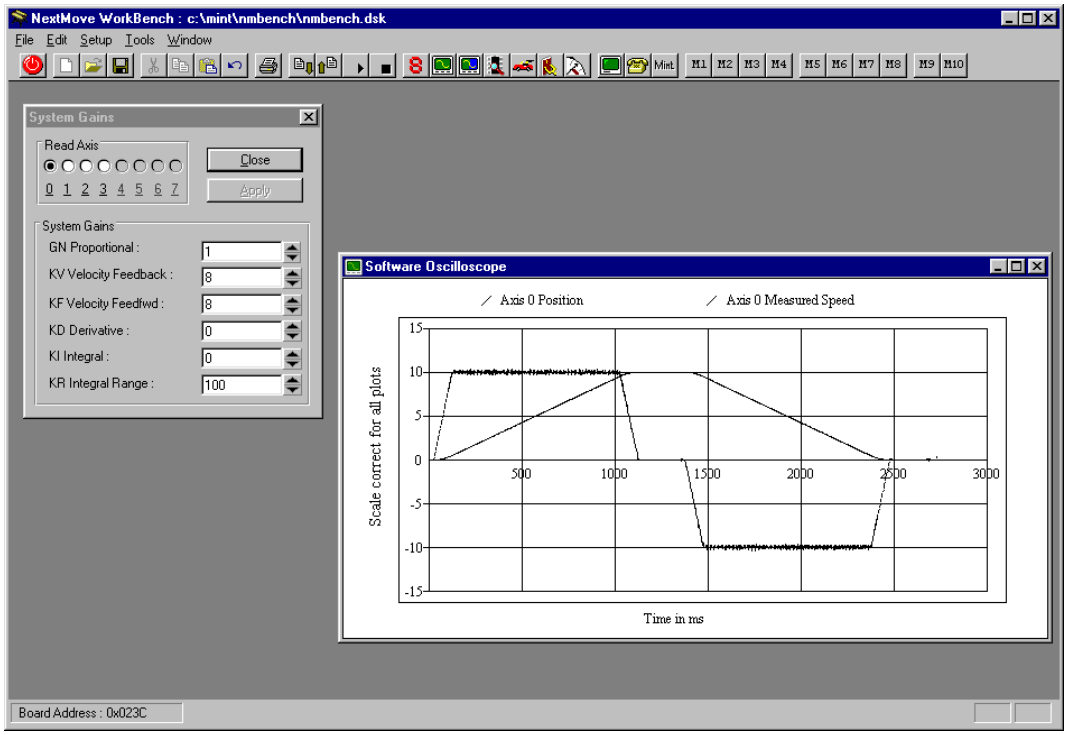


Figure 25: Fine tuning system gains

See section 12.13 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.

5.4.16 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.

5.5 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 18).

5.5.1 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 5.6.

5.5.2 Pulse Generator Outputs

The controller has three outputs for each of four possible stepper motors - **PULSE**, **DIR** and **BOOST**. These are referenced 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. The stepper signals are shown in the table.

Pin	Signal	Function
37	**pulse0**	Pulse Output 0
38	**pulse1**	Pulse Output 1
39	**dir0**	Direction Output 0
40	**dir1**	Direction Output 1
41	**boost0**	Boost Output 0
42	**boost1**	Boost Output 1
43	**gnd**	Digital Ground
44	**gnd**	Digital Ground
45	**pulse2**	Pulse Output 2
46	**pulse3**	Pulse Output 3
47	**dir2**	Direction Output 2
48	**dir3**	Direction Output 3
49	**boost2**	Boost Output 2
50	**boost3**	Boost Output 3
51	**gnd**	Digital Ground
52	**gnd**	Digital Ground

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).

5.5.3 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. The signals are found on JP6 on the breakout board.

Pin	Signal	Function
53	**relay-com**	Error Relay Common Connection
54	**relay-nc**	Error Relay Normally Closed Connection
55	**relay-no**	Error Relay Normally Open Connection

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:

2. 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.

3. 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.

5.5.4 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 18.

5.5.5 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.

5.6 Input/Output

The *NextMove PC* controller brings out all Input/Outpu“ 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 8.

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 ±2.5V. In addition the range of each input can be extended by a factor of four (using jumpers) to ±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.

5.6.1 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

5.6.2 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 stepper 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 **ERRORDECEL** 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..11	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.

Introduction to the MINT™ Programming Language

6

MINT™ is the programming language used to program the controller in order to meet the requirements of specific applications. Chapter 3 introduced MINT; this chapter provides a more detailed look at the language with example applications.

- ◊ Creating a MINT configuration file
- ◊ Look at a cut to length, XY table and product infeed applications.
- ◊ Introduction to the MINT comms array for data transfer

MINT is the programming language used to program the controller in order to meet the requirements of specific applications. MINT provides control of the I/O and motion control aspects of the controller using Basic-like programming structures and keywords. There may be an application where a thumbwheel switch (fed into the digital inputs of the controller) sets a distance to move and a potentiometer (fed into one analog input on the controller), changes the slew speed. A simple MINT program can be written to achieve this.

MINT programs consist of two files. The *Configuration buffer* stores information relating to the machine set-up, for instance the servo loop gains. The *Program buffer* stores the actual motion control program. In fact the two files are the same and can contain the same instructions, except that the *Configuration buffer* is only 8K maximum size, while the *Program buffer* can be up to 100K. Additionally, there is a third method of storing information in the controller - in the form of array data - which can be used as variables within a program. More information is provided in *MINT for NextMove Programmers Manual*.

6.1 The Configuration File

The *Configuration buffer* is used to store appropriate defaults for a particular system. Once gains and speeds have been found, these should be incorporated into a *Configuration buffer* for the system.

Whenever the program is **RUN**, the *Configuration buffer* is executed first. The following provides a list of parameters necessary for set-up:

- **Servo loop gains** - to tune the system response.
- **Scale factor** - to set the units of measure of the application. Refer to *MINT for NextMove Programmers Manual* for details of using **SCALE**.
- **Maximum following error (MFOLERR)** - to set a safe maximum difference between actual and desired positions.
- **Default speeds (SPEED), accelerations (ACCEL) and decelerations (DECEL) for the system** - to determine the shape of the trapezoidal velocity profile (**RAMP**).
- Error features such as hardware and software limits (**FWLIMITIN, FWSOFTLIMIT, REVLIMITIN, REVSOFTLIMIT**), stop inputs (**STOPINPUT**) and error inputs (**ERRORIN**).

These parameters are generally to be set-up only once for an application, but can at any time be altered in the *Program buffer*.

A typical *Configuration buffer* for a five axis system is:

CONFIG.CFG

```
REM CONFIG.CFG
REM Baldor Optimised Control

RESETALL                : REM to ensure previous setting cleared
CONFIG[0,1,2,3] = _servo, _servo, _servo, _off
CONFIG[4,5,6,7] = _stepper, _stepper, _off, _off

AXES[0,1,2,4,5]         : REM 5 axis of motion

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

REM system gains
GAIN[myservo]    = 10;
KVEL[myservo]    = 40;
KVELFF[myservo] = 40;   :REM assuming a torque amplifier
DGAIN[myservo]   = 0;
KINT[myservo]    = 0;

REM position/SPEED parameters for servos
SCALE[myservo] = 2000;  : REM units revs (500 line encoder)
SPEED[myservo] = 60;    : REM max SPEED 60 revs/sec
ACCEL[myservo] = 150;   : REM max accel 150 revs/sec^2
DECEL[myservo] = 150;   : REM max decel 150 revs/sec^2
ERRORDECEL[myservo] = 300; : REM error deceleration rate
RAMP[myservo] = 0;      : REM Trapezoidal motion
MFOLERR[myservo] = 1;   : REM 1 rev maximum following error
FEMODE[myservo] = 1;    : REM enable following error on all servos

REM position/SPEED parameters for steppers
SCALE[mystepper] = 1000;    : REM units revs
SPEED[mystepper] = 10;      : REM max SPEED 10 revs/sec
ACCEL[mystepper] = 25;      : REM max accel 25 revs/sec^2
DECEL[mystepper] = 25;      : REM max decel 25 revs/sec^2
ERRORDECEL[mystepper] = 300; : REM error deceleration rate
RAMP[mystepper] = 0;        : REM Trapezoidal motion

REM set input triggering
INACTIVEMODE =  0xF80000    : REM bits/inputs 19 - 23 edge triggered
POSEDGEIN = 0xF80000        : REM bits/inputs 19 - 23 RISING triggered
NEGEDGEIN = 0x000000        : REM no Falling edge triggered

REM set safty features
REVLIMITIN[myservo] = 0,2,4 : REM reverse limit switch
FWLIMITIN[myservo] = 1,3,5  : REM forward limit switch
REVLIMITIN[mystepper] = 6,7 : REM reverse limit switch
FWLIMITIN[mystepper] = 6,7  : REM forward limit switch
HWLIMIT = 3;                : REM define action on hardware limit
ENLIMIT                     : REM enable generation of hardware limits

HMINPUT[myservo] = 8,9,10   : REM home input switch
```

```
HMINPUT[mystepper] = 11;     : REM home input switch

STOPINPUT[myservo] = 12;     : REM stop input switch
STOPINPUT[mystepper] = 13;   : REM stop input switch
STOPMODE[myservo] = 1;
STOPMODE[mystepper] = 1;

ERRORIN[myservo] = 14;       : REM Error Input switch
ERRORIN[mystepper] = 15;     : REM Error Input switch

ENOUTPUT[myservo] = 0,1,2    : REM drive enables
ENOUTPUT[mystepper] = 3;     : REM drive enables

RESETALL                     : REM enable all drives
REM end of file
```

File is in the MINT\NMPC\EXAMPLES directory.

In the above example, the default axis list is 0,1,2,4,5 set-up using the **AXES** keyword. The semi-colon is used to apply the parameters to all axes specified in square brackets (**[]**) , if the square brackets are omitted then the semi-colon will cause the command to reference all axes in the axes string. Refer to *MINT for NextMove Programmers Manual* for further details on program language syntax.

Please note that the values for gains, speeds etc. are given only as an example. It is up to the user to determine the best values for a system. Failure to do so may result in damage to the machine.

6.2 The First MINT Program

This first MINT program example consists of a simple *Configuration* and *Program buffer* which is used to index a motor by a set distance entered via the terminal. The Configuration and *Program buffer*s can be found in the **\MINT\NMPC\EXAMPLES** directory.

The *Program buffer* is called **FIRST.MNT** and the *Configuration buffer* is called **FIRST.CFG**. The example has been written for a single axis of motion, connected to axis 0 of the controller.

FIRST.CFG

```
REM File name: FIRST.CFG
REM Baldor Optimised Control
REM Configuration file for first MINT program
```

```
AXES[0]         : REM This program only uses axis 0 (the first axis)
RESETALL

SCALE = 2000    : REM Scale factor for revs - 500 line encoder (500*4)
GAIN = 1        : REM Servo loop gains, these should be changed ..
KVEL = 5        : REM .. to the values found during servo set-up
DGAIN = 0
KINT = 0
KVELFF = KVEL

SPEED = 50      : REM Default speed during positional moves 50 rev/s
ACCEL = 200     : REM 200 rev/s^2
DECEL = 200     : REM 200 rev/s^2

REM end of file
```

File is in the MINT\NMPC\EXAMPLES directory.

Any characters after a **REM** (remark) statement are ignored, which allows comments to be inserted into the program to make it more intuitive. It is important that back-up copies of *Program* and *Configuration buffers* are kept on a disk. The first line of this program indicates the name given to the disk file.

It must be remembered that the gains and scale factor may have to be changed to suit a particular motor.

FIRST.MNT

```
REM File name: first.mnt
REM Baldor Optimised Control
REM Program to demonstrate use of the keypad to cause
REM the motor to repeat an index distance ten times

REM initialise variables used in program
index_length = 0

LOOP

 CLS : REM clear screen

 REM Print up screen requesting user to enter move distance
 LINE 1,"Enter index distance"

 REM input statement uses formatted input xx.x
 LOCATE 5,2 : REM put cursor at column five line 2
 INPUT index_length USING 2,1

 REM Perform move ten times printing cycle number on screen

 LINE 1,"Indexing"
 FOR cycle = 1 TO 10
  MOVER = index_length  REM move relative command
  GO                    REM start motion
  LINE 3,"Cycle no: ",cycle;
  PAUSE IDLE
 NEXT

ENDL : REM go back to start of loop for next motion

END  : REM end of program
REM end of file
```

File is in the MINT\NMPC\EXAMPLES directory.

The *Program buffer*, **FIRST.MNT** should be downloaded to the controller and **RUN** typed at the > prompt in the terminal screen.

Tip: If during program execution the controller aborts and prints the error message:

```
ERROR: Program: Following error at line xx on axis 0
```

this probably signifies that the motor is not correctly set-up. It should be verified that the configuration is in accordance with the instructions in section 5.

6.2.1 Program Narrative

The first MINT statement in the file (which is not a comment) is the line:

```
index_length = 0
```

this defines a variable called **index_length** and initializes it to the value 0. **index_length** is used later in the program to store the length of move entered by the operator.

The **LOOP** statement in the program signifies the start of a loop from which the program never exits. It simply marks the point to which the program jumps when it encounters the **ENDL** statement.

After clearing the terminal screen of any erroneous information printed by previous programs (**CLS**) the statement:

```
LINE 1,"Enter Index Distance"
```

prints a message on line one of the terminal. The **LINE** keyword enables easy location of text on displays. Note that it is equally possible to use the **LOCATE** and **PRINT** statements to do this:

```
LOCATE 1,1
PRINT "Enter index distance”
```

The **LINE** keyword however, blanks the first 20 characters from the left before printing any text, ensuring that there are no characters left on that line from previous **PRINT** statements. This makes it ideal for use with CAN keypad nodes (see section 10).

The next statement requests an input from the operator, being the number of revolutions of the motor wanted to index. The **USING** statement is used here to print the number in a set format. In this case two integer characters followed by a single decimal character. (Note that the **SCALE** keyword in the *Configuration buffer* has been set-up so that all distances and speeds are in revolutions of the motor).

The desired length can be entered by typing in the number of revolutions on the keyboard. Pressing return will cause the program to move the motor the set distance ten times.

The motor movement is achieved by the statement:

```
MOVER = index_length
GO
```

MOVER is a relative positional move. It causes the motor to move the number of revolutions specified, by using the accelerations and speed set-up previously in the *Configuration buffer*. The **GO** command is required to actually start the motion. This is beneficial if synchronizing both relative and absolute moves on two or more motors is necessary. **GO** is not required for certain types of moves, notably continuous speed control using the **JOG** command.

It can be seen that the **MOVER** statement is surrounded by a **FOR .. NEXT** loop statement. This causes the statements inside the loop to be executed 10 times with the number being stored in the variable *cycle* each time. This is printed each time the loop is repeated.

Finally, the program arrives at the **ENDL** statement. This causes it to jump back to the **LOOP** statement so that the next index length can be entered. **FOR .. NEXT** and **LOOP .. ENDL** are two examples of the four different loop statements. These statements are useful for writing machine control programs.

To end program execution, the keys **[Ctrl]+[E]** (**Ctrl** and **E** together) should be typed at the terminal screen. The prompt should be displayed.

6.3 A Simple Cut to Length Feeder

In many applications, MINT can be used to program the system as a stand-alone machine controller. This program illustrates such a program for a simple cut-to-length machine. The program allows the operator to enter a product length, feed speed and number of feed cycles. It will then record the total length of product that has been cut.

If the controller has been set-up according to the instructions in the first part of this manual and the standard controller is being used, then it should be possible to get this program operating within a few minutes, by using one or two motors with their shafts in free air.

The *Program buffer* **FEEDER.MNT** and the *Configuration buffer* **FEEDER.CFG** will have been installed in the appropriate directory. *NextMove WorkBench* should be started and these files downloaded to the controller as before. **RUN** should be typed at the **P>** prompt.

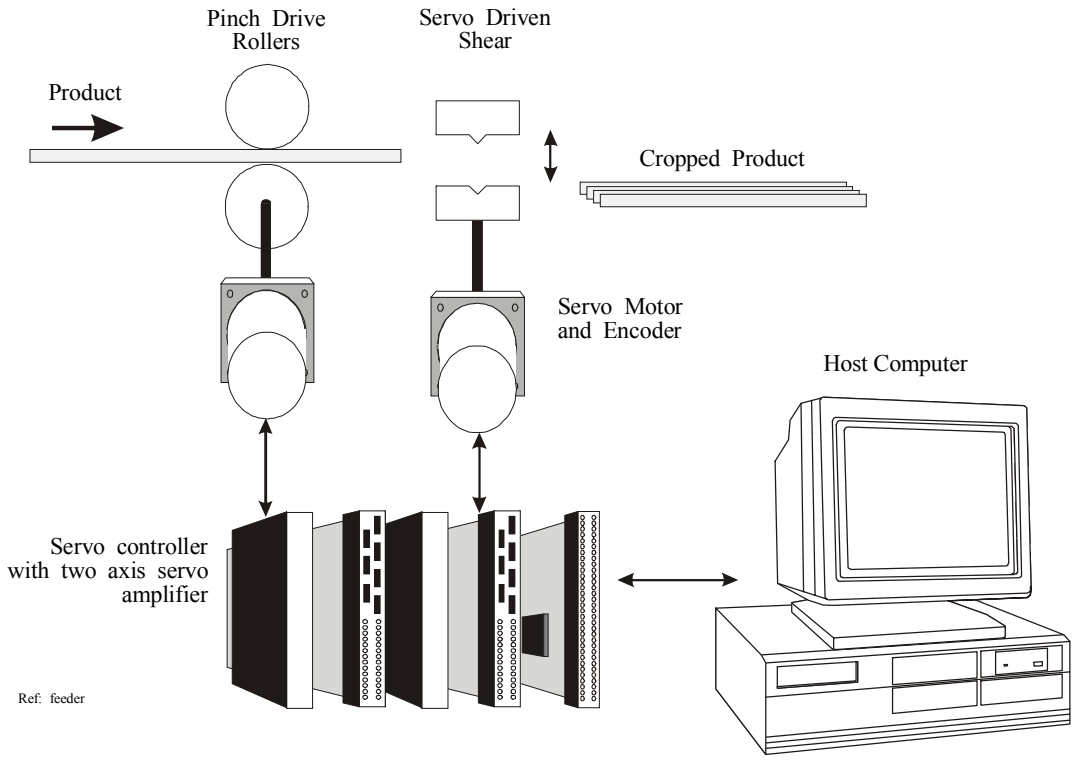


Figure 26: Schematic Diagram of the Cut to Length Feeder

6.3.1 *Configuration Buffer* FEEDER.CFG

FEEDER.MNT

```
REM File name: FEEDER.CFG
REM Baldor Optimised Control
REM Configuration file for simple cut to length machine

CLS             : REM clear screen
PRINT "Setting up axes, please wait...",

AXES[0,1]
RESETALL

SCALE = 50;     : REM 50 encoder counts = 1 mm on this application
GAIN = 0.5;     : REM Servo loop gains
KVEL = 5;
KVELFF = KVEL;
KINT = 0;
SPEED = 4000;   : REM 4000 mm/s
ACCEL = 40000;  : REM 40000 mm/sec^2
DECEL = 40000;  : REM 40000 mm/sec^2
RAMP = 0;       : REM no 's' ramping
STOPINPUT = 0;  : REM set stopswitch on axes 0 and 1 to be i/p zero
STOPMODE = 1;   : REM in event of a stop input, stop motion and
                  REM call stop handler

PRINT "OK"
REM end of file
```

File is in the MINT\NMPC\EXAMPLES directory.

The *Configuration buffer* contains information specific to the servo system set-up. Detailed information of the MINT command syntax is given in *MINT for NextMove Programmers Manual*.

On running the program and configuration files the message:

```
"Setting up axes, please wait... "
```

will be displayed on the screen during the execution of the *Configuration buffer*. Once the axes have been setup the controller displays

```
"OK"
```

following which the controller compiles and executes the *Program buffer*.

The statement **`AXES[0,1]`**, indicates that the controller is fitted with two axes of motion and that all commands thereafter will relate to these two axes unless explicitly indicated by enclosing the axis number in brackets. For example, the command:

```
SPEED[1] = 10
```

sets the speed of axis **1** to **10**, but

```
SPEED = 10;
```

sets the speed of both **0** and **1** to **10**. Note that the use of the semi-colon to set both axes to **10**, otherwise it would be necessary to type **SPEED = 10,10**.

Also note the inclusion of the **RESETALL** command. Although not strictly necessary, this guarantees that the controller starts in a known state with all error flags reset to their default values and position set to zero.

The code **SCALE = 50**; sets the system units in relation to the number of encoder quadrature counts. If there were 50 counts per mm of linear movement, and it was desired to program speeds and distances in mm, **SCALE = 50** would be set .

The remainder of the file contains system gains and configuration information.

These values may have to be changed to achieve a stable system according to the particular motor/drive being using - refer to section 5.4.10, Setting System Gain of this guide for further information.

6.3.2 *Program buffer* FEEDER.MNT

```
REM Program to demonstrate user interface and basic motion commands
REM using a 'cut to length machine'

RESETALL          : REM reset all motion parameters to default values
GOSUB initialise  : REM call initialise subroutine
GOSUB main_loop   : REM call main subroutine
END

#non_volatile
REM dummy definitions of variables stored in non-volatile RAM
REM this routine is not actually called and therefore the variables
REM are defined but not initialised to zero so that thier programmed
REM values are retained
 cycles = 0       : REM number of material feed cycles
 slew_speed = 0   : REM speed of material feed
 length = 0       : REM amount of material feed
RETURN

#initialise
  REM This subroutine sets up various parameters when program starts
  SPEED = slew_speed : REM restore speed stored in non-volatile memory
  count = 0          : REM total length of material fed
  jog_sp = 2000      : REM default jog speed in manual mode
```

```
RETURN

#main_loop
REM This subroutine is the main program loop, it prints the start-up
REM screen and handles operator selections by calling further
REM subroutines
  LOOP
    REM Print up menu screen on four line operator display
    REM Line 4 is the 'soft keys' for operator selection
    CLS
    LINE 1,""
    LINE 2,"XYZ Widget Company"
    LINE 4,"Main Menu"
    LINE 6,"1. Start Motion"
    LINE 7,"2. Setup Parameters"
    LINE 8,"3. Manual Override"
    LINE 9,"4. EXIT"
    LINE 11,"Enter Option : ",
    REM Read the key pressed to the variable 'key'. Test to see if
    REM is either 1, 2, 3 or 4
    key = 0
    REPEAT
      key = INKEY
    UNTIL key > '0' & key < '5'
    BEEP
    IF key = '1' THEN GOSUB start : REM call start subroutine
    IF key = '2' THEN GOSUB setup
    IF key = '3' THEN GOSUB manual
    IF key = '4' THEN GOSUB quitProgram
  ENDL
RETURN

REM This subroutine is allows movement; roll back & forth
#manual
  CLS
  LINE 1,""
  LINE 2,"XYZ Widget Company"
  LINE 4,"Manual Override Menu"
  LINE 6,"1. Fast motion"
  LINE 7,"2. Slow motion"
  LINE 8,"3. EXIT"
  LINE 9,"                "
  LINE 10,"Enter Option : ",
  REM Print new menu where soft keys are FAST motion, SLOW motion and
  REM EXIT to main menu
  jog_sp = 2000
  REM read key presses and set fast or slow motion or exit
  key = 0
  REPEAT
    key=INKEY
  UNTIL key > '0' & key < '4'
  BEEP
  IF key = '1' THEN jog_sp = 2000 : REM fast motion
```

```
  IF key = '2' THEN jog_sp = 100  : REM slow motion
  IF key = '3' DO
    key = 0
  ELSE
    LOOP
      REM Moves the motor back and forth using the arrow keys < and >
      REM space bar stops motion
      LINE 6,"Left arrow  to move left"
      LINE 7,"Right arrow to move right"
      LINE 8,"Space bar   to stop"
      LINE 9,"Enter       to EXIT"
      LINE 10,"                "
      LINE 11,"Enter Option : ",
      REPEAT
        key = INKEY
      UNTIL key = 15 OR key = 16 OR key = 32 OR key = 13

      IF key = 15 DO
        JOG = -jog_sp
      ELSE
        IF key = 16 DO
         JOG = jog_sp
        ELSE
          IF key = 32 DO
            STOP
          ELSE
            STOP : EXIT
          ENDIF
        ENDIF
      ENDIF
    ENDL
  ENDIF
RETURN

REM Subroutine to allow operator to set-up product length, feed speed
REM and number of repetitions
#setup
  LOOP
    CLS
    LINE 1,""
    LINE 2,"XYZ Widget Company"
    LINE 4,"Setup Parameters Menu"
    LINE 6,"1. Set Length"
    LINE 7,"2. Set Speed"
    LINE 8,"3. Set Number of Cycles"
    LINE 9,"4. EXIT"
    LINE 11,"Enter Option : ",
    key = 0
    REPEAT
      key = INKEY
    UNTIL key > '0' & key < '5'
    BEEP
    IF key = '1' THEN GOSUB get_len
```

```
    IF key = '2' THEN GOSUB get_sp
    IF key = '3' THEN GOSUB get_cy
    IF key = '4' THEN key = 0 : EXIT
  ENDL
RETURN


REM subroutine to get length of material
#get_len
  LINE 13,"Enter index distance :        mm"
  REM formated input XXX.X
  LOCATE 24,13 : INPUT length USING 3,1
RETURN

REM get number of feed cycles, up to 99 repetitions
#get_cy
  LINE 13,"Enter number of indexes required : "
  LOCATE 36,13 : INPUT cycles USING 3
RETURN

REM subroutine to get speed, this example validates entered number to
REM make sure that it is less than 4000 and greater than 100
#get_sp
  REPEAT
    LINE 13,"Enter maximum slew speed :      mm/s"
    LINE 14,"Range 100-4000mm/s",
    number = slew_speed       : REM store before validating
    LOCATE 28,13 : INPUT number USING 4
  UNTIL number >= 100 AND number <= 4000
  slew_speed = number : REM if valid update variable and SPEED command
  SPEED = slew_speed
RETURN

REM Run automatic cycle
#start
  CLS
  LINE 1,""
  LINE 2,"XYZ Widget Company"
  LINE 4,"Machine Running ... press any key to STOP"
  LINE 6,"Cycle no :"
  LINE 7,"Material :"
  REM FOR .. NEXT loop executes n times where n = cycles
  FOR index = 1 TO cycles
    REM index material - move axis 0 distance given by length
    MOVER[0] = length : GO[0]
    PAUSE IDLE  : REM wait for prevoius move to finish
    REM perform punch operation by moving axis 1 up and down
    MOVEA[1] = 10 : GO[1]
    MOVEA[1] = 0 : GO[1]
    BEEP
    count = count + length : REM accumulate material fed
    REM print status information
    LOCATE 12,6 : PRINT index USING 2;
```

```
    LOCATE 12,7 : PRINT count USING 5,1;
    IF INKEY THEN STOP : EXIT : REM if a key is pressed then stop
  NEXT  REM end of FOR .. NEXT loop
  REM End of programmed number of cycles
  LINE 4,"Machine Stopped ... press any key to EXIT"
  PAUSE INKEY
  BEEP
RETURN

#quitProgram
  CLS                    : REM clear screen
  PRINT "Program ended"
  END                    : REM end program
return

REM Error handling subroutine called by system in the event of
REM excessive following error (machine jam) or limit switch error
#onerror
  CLS
  LINE 1,"***************** Error ******************"
  LINE 2,"*** Machine JAM press any key to RESET ***"
  LINE 3,"*********** and RE-RUN program ***********"
  PAUSE INKEY
  CANCEL[0,1,2,3] : REM cancel error
  RESETALL
  RUN
RETURN

REM Error handling routine called by system when STOP input
REM (guard switch) is asserted.
REM Subsroutine prints message on operator screen, then
REM returns to main program
#stop
  LINE 15, "*** Motion stopped, GUARD OPEN ****"
  PAUSE STOPSW = 0 : REM wait for guard to be closed
  LINE 15, "                                    "
RETURN

REM end of file
```

File is in the MINT\NMPC\EXAMPLES directory.

6.3.3 Cut To Length Program Narrative

The program has been written in a structured method for ease of maintenance, according to the following prototype:

```
REM program starts here
GOSUB init
GOSUB main
#init
```

```
...
RETURN

#main
...
RETURN
```

The program is well commented and by making suitable adjustments to the **SCALE** factor and gains in the *Configuration buffer* it should be possible for the program to operate on any system equipped with at least one axis of motion.

The **#non_volatile** subroutine is defined but never called. This is used to define some values which are used in the program to store the product length entered by the user, for example. Because the subroutine is never called they are never actually set to zero on power-up and therefore the last values entered by the user are retained.

The initialization sub-routine, #**init** is called first. Initialization sets up some defaults for the system.

The **RETURN** statement causes the program to jump back to the statement directly after **GOSUB init** (the main loop) is called.

The main loop simply prints up the following message on the display:

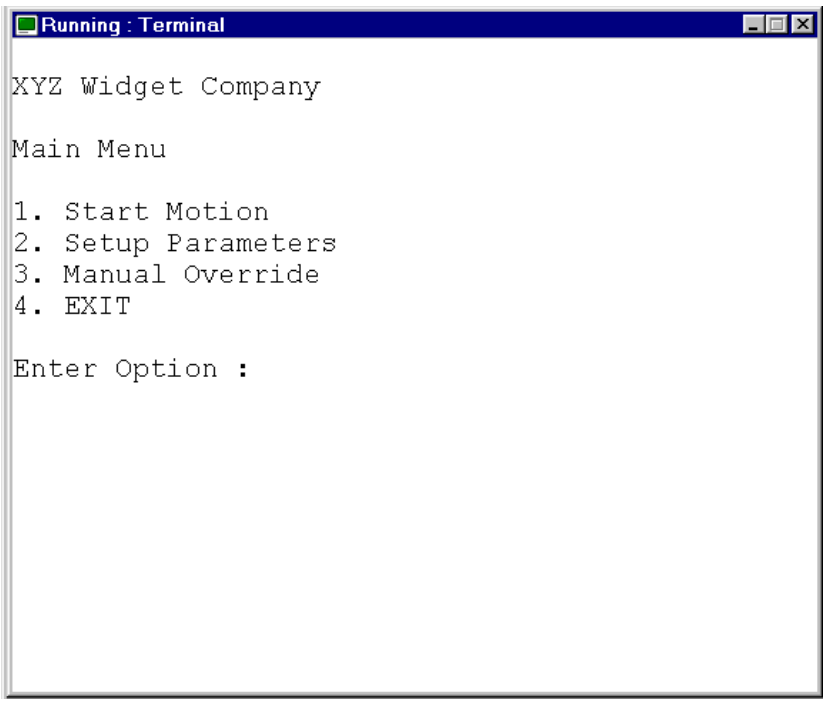


Figure 27: Main Menu

This main program loop displays the text to the terminal and checks to see whether a key has been pressed at the keyboard. When a valid key is pressed (1, 2, 3 or 4) the appropriate subroutine is called. Note the use of **INKEY** to read the value of a key pressed.

Pressing **1** will start the operation. **2** sets up the parameters for speed, length and number of cycles. **3** enters manual mode, allowing the user to move the material backwards and forwards and **4** exits the program returning to the command prompt.

Manual mode uses the **JOG** command to move the axis in a positive or negative direction. This is command is designed for continuous speed control, using this command the desired speed can be modified and the **ACCEL** and **DECEL** rates are used to fade the speed changes together.

Pressing **2** will enter set-up mode, and the following display shown:

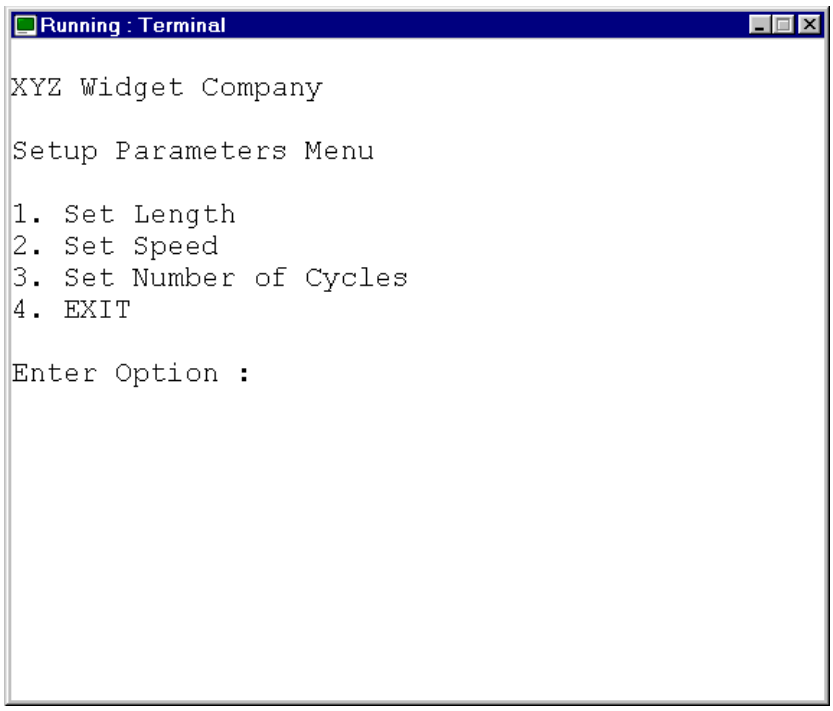


Figure 28: Set-up Menu

Pressing **1**, **2** or **3** calls the **#get_len**, **#get_cy** and **#get_sp** subroutines which allow the operator to enter data. The **INPUT USING** command is used to provide formatted input. In this case the number is displayed as three integers.

The **#start** routine contains the code to start movement. In this example, a relative move (distance from start) is executed on the material feed axis; followed by an absolute (relative to a fixed zero position) up-down movement on the press axis. Note that in both the case of **MOVER** (move relative) and **MOVEA** (move absolute) the **GO** command is used to start motion.

The move commands are surrounded by a **FOR - NEXT** loop which causes them to be repeated a number of times specified by the variable **cycles**, which is entered by the user during the set-up routine.

The **#ONERROR** routine near the end of the program is a routine which is called in the event of a controller error, such as when the motor jams or a limit switch is hit. In this case, the routine prints an error message and waits for a key to be pressed before re-running the program from the start.

The program can be tried and some of the parameters at the operator keypad changed. Program execution can be aborted by pressing **[CTRL]+[E]** at the terminal screen.

6.3.4 Using Batch Numbers

A common requirement in a cut-to-length machine is to store the parameters (speed, length and number of cycles etc.) that have been entered for different products and restore these quickly using a batch code. MINT allows this by using array variables and making the index to the array a batch code. More information on array variables is given in *MINT for NextMove Programmers Manual*.

The program **FEEDER2.CFG** is an example of this type of program. The variables: **slew_speed**, **length** and **cycles** are redefined to be arrays each of length 99 by using the **DIM** statement:

```
DIM slew_speed(99)
DIM length(99)
DIM cycles(99)
```

A fourth option is added to the main menu allowing the entry of a batch number into the variable **batch** which is then used whenever a reference is made to the above variables. Therefore the move statement becomes:

```
MOVER = length(batch)
```

where **batch** is the index to the array called **length**.

Another advantage of using arrays to store operator input information is that they can be uploaded into a computer and stored to disk in a similar manner to *Program* and *Configuration buffers*. Similarly a file can be created which contains the data for each batch number and this can be downloaded to the controller. This is all achieved, while the program on the control system is running by using the **LOAD** and **SAVE** commands.

6.4 X-Y Teach and Replay Program

This example program illustrates the use of array variables for storing and replaying positional data programmed by the operator. This program records ten x,y data points and then replays them printing the X Y position on the display as it does so. The *Configuration buffer* **TEACH.CFG** is used for the application. In order to run this program, it is not necessary to have an XY table - two motors will work just as well.

TEACH.CFG

```
REM TEACH.CFG
REM Baldor Optimised Control
REM Config file for XY table and keypad node

RESETALL       : REM to ensure previous setting cleared
AXES[0,1]      : REM 2 axis servo system

REM system gains
GAIN   = 10;
KVEL   = 40;
KVELFF = KVEL; :REM assuming a torque amplifier
DGAIN  = 0;
KINT   = 0;

REM position/SPEED parameters
SCALE = 2000;  : REM units revs (500 line encoder x 4)
SPEED = 60;    : REM max SPEED 60 revs/sec
ACCEL = 150;   : REM max accel 150 revs/sec^2
DECEL = 150;   : REM max decel 150 revs/sec^2
RAMP = 0;      : REM Trapezoidal motion
MFOLERR = 1;   : REM 1 rev maximum following error

REM setup keypad node
NODETYPE.14 = _keypad   : REM add keypad node to CAN network
PAUSE NODELIVE.14       : REM Wait for node to become live
KEYPADNODE._tmLCD1 = 14 : REM enable keypad node

REM end of file
```

File is in the MINT\NMPC\EXAMPLES directory.

This configuration file includes the setup of a CAN keypad node which is used in the program **TEACH.MNT** to control the movement of the XY table.

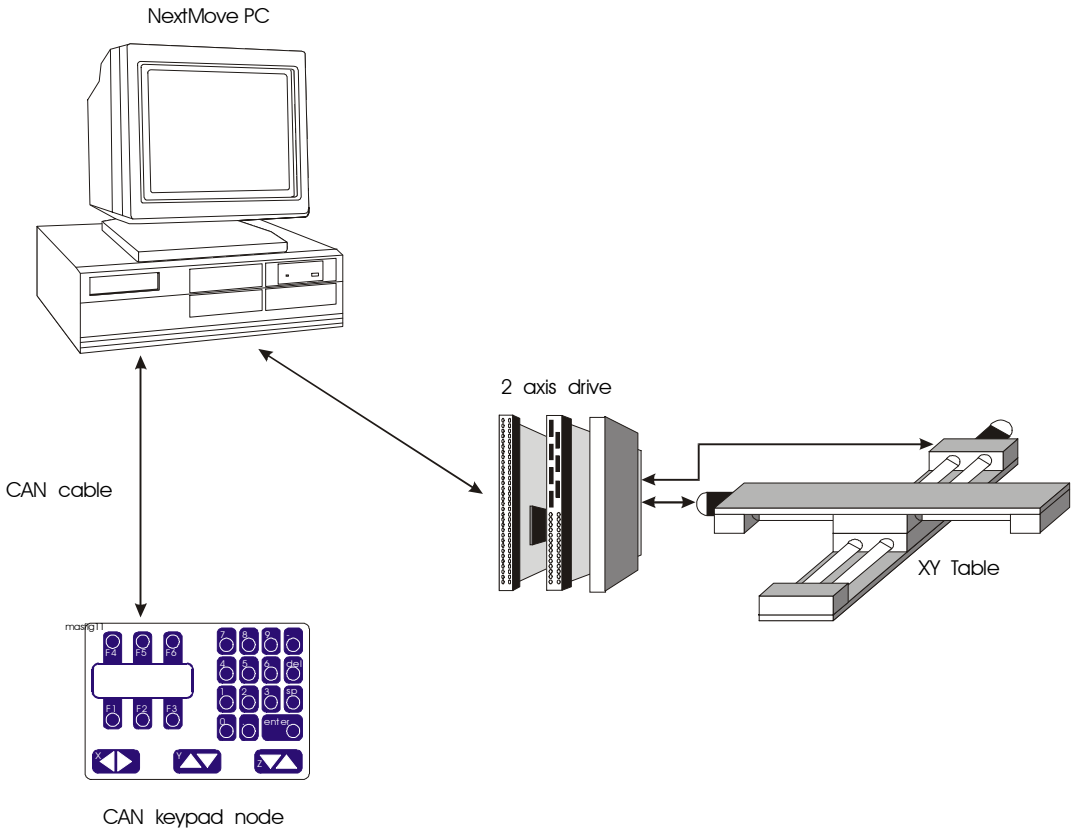


Figure 29: XY Table Control System

TEACH.MNT

File is in the MINT\NMPC\EXAMPLES directory.

This program uses the **READKEY** function to jog the motors X and Y back and forth in response to the pressing or activation of the *Keypad*Node **X** and **Y** keys. It possible however, to adapt the program to work with a standard potentiometer joystick connected to the analog inputs. The analog inputs return a 12 bit value 0 - 4095 between -10V and +10V (or 0-5V). The optimum way to connect the joystick is to connect +/-12V across the potentiometer and connect the wiper to the analog inputs **1**+ and **2**+. The differential inputs **1**- and **2**- should be connected to analog ground.

The analog inputs, read by the keywords **ANALOGUE1** and **ANALOGUE2** (abbreviated to **A1** and **A2**), will return 0 when the joystick is in the fully negative position and 4095 when in the fully positive position. A value of 2048 will be returned when the joystick is in the middle position. Most joysticks are equipped with trimmers to adjust the zero position, but electrical noise can cause some jitter about 0V, so it is advisable to employ a deadband of 10 counts around the zero point.

The following code fragment illustrates the implementation of analog joystick control, which would replace the **REPEAT ..UNTIL** loop in **TEACH.MNT**:

```
REPEAT  REM repeat until record button F1 is pressed
  REM jog at X,Y speed given by analog inputs, range ..
  REM is 0-4095, subtract 2048 to give bi-directional control
  jog_x = A1-2048
  jog_y = A2-2048
  REM deadband of 10 points either side of zero
  IF ABS(jog_x) < 10 THEN jog_x = 0
  IF ABS(jog_y) < 10 THEN jog_y = 0
  JOG = jog_x, jog_y
UNTIL INKEY = 'a'
STOP[0,1] REM stop jog motion
```

Note that this example provides variable speed movement on the table depending on the position of the joystick, and also allows simultaneous movement on X and Y, which is not possible using the operator panel.

Another worthwhile feature would be to home the axes during power-up of the system. This can be achieved by using home switches on the table and the **HOME** keyword. Further details of this are given in *MINT for NextMove Programmers Manual*.

6.5 Software Gearbox Example - Coil Winding Machine

The controller has a *software gearbox* function making it ideal for controlling machines that have to follow a master encoder or pulse train. A typical application is to wind wire onto a drum rotating at a speed dictated by a mechanical drive from a wire drawing machine. When the spool is empty, the spool rotates at the highest speed, slowing down as layers of wire are wound onto the spool. The thickness of wire, width of the spool, and spacing between each wire must be changed to cope with different thickness of product.

An encoder on the spool rotation provides a speed signal. The controller uses this signal as a reference for a servo motor which drives the spool back and forth via a lead screw.

An operator panel on the front of the machine allows the operator to change *the following ratio* (spacing between wires) and the position of the left and right end of travel.

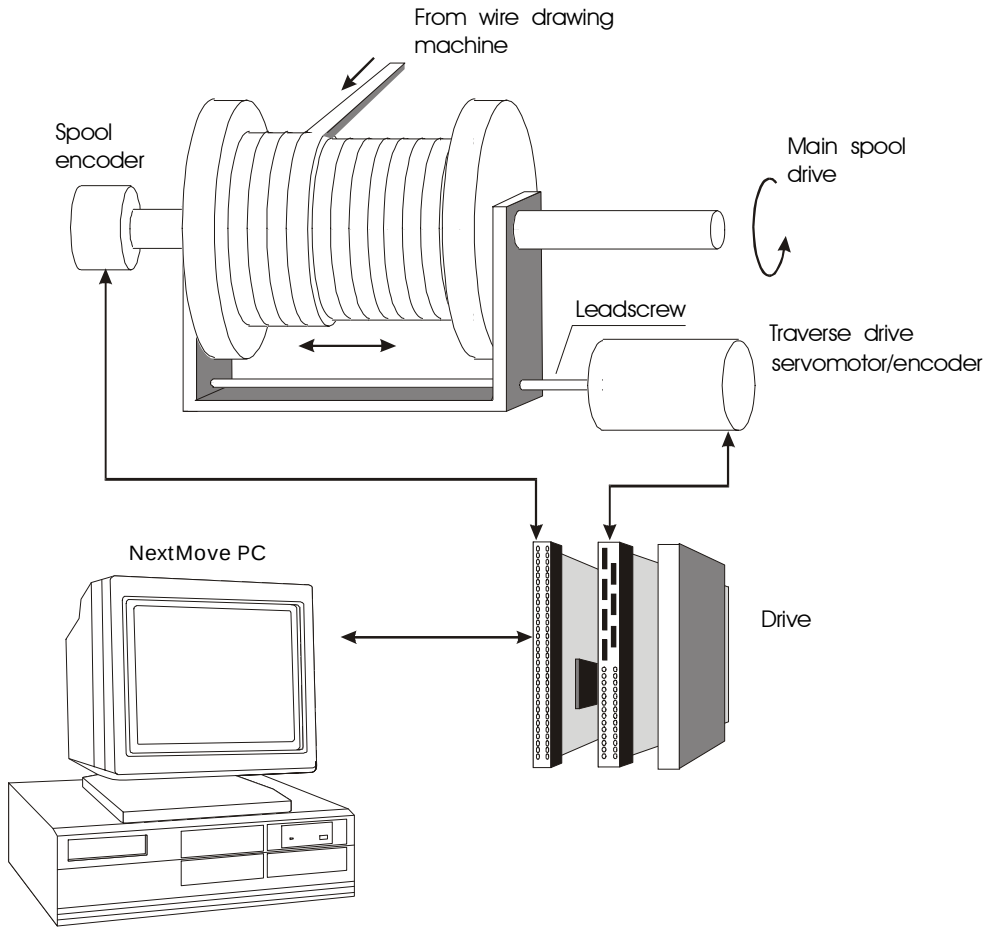


Figure 30: General Layout of Coil Winding Machine

SPOOLER.MNT

File is in the MINT\NMPC\EXAMPLES directory.

6.5.1 Program Narrative

The variable **ratio** is used to store the 'software gearbox' ratio. That is, the number of servo motor encoder pulses moved per pulse from the spool encoder. This ratio is variable and can be adjusted at any time. **left_limit** and **right_limit** store the limits of linear travel. In this example, there are 4000 quadrature encoder counts per revolution of the servo motor encoder and a 4mm pitch lead screw. Therefore, setting the keyword **SCALE** to 1000 means that the limit positions may be expressed in mm.

This program is an example of a spooler with the display simply telling the user basic information about the state of the program. A typical enhancement on practical machines is a 'dwell on reversal' to allow the material to ride up onto the new layer.

6.5.2 Remote Operation Using the COMMS array

In many applications the spooler system may be required to be adjusted remotely, under instruction of a host computer and perhaps while the spooling operation is in progress. The controller's protected communications protocol allows data to be exchanged between a host computer and the controller by means of a 99 location array called **COMMS**. **COMMS** operates like a dual port RAM; the host computer can write to and read the value of specific locations, over the serial port, for use by the controller program. This happens automatically, without any overhead in the MINT program itself.

For remote operation, the variables **ratio**, **left_limit** and **right_limit** would be replaced by three **COMMS** letterbox addresses:

```
REM routine to perform spooling
REPEAT
  REM forward direction
  FOLLOW = COMMS(1)  : REM COMMS(1) = ratio
  PAUSE POS > COMMS(2) : REM COMMS(2) = right limit
  REM reverse direction
  FOLLOW = -COMMS(1)
  PAUSE POS < COMMS(3) : REM COMMS(3) = left limit
UNTIL INKEY = 13
```

If a value is changed by the host computer, this automatically appears in the program while it is running. A protected protocol is used to make the system fault tolerant. This protocol is ANSI 3.28 based, compatible with many other manufacturers of drive systems. The structure and implementation of protected communications are discussed in detail in *MINT for NextMove Programmers Manual*. These cover the use of *NextMove WorkBench* for reading and writing to the controller using protected communications.

6.6 Infeed Packaging Machine

MINT offers a number of features to facilitate the control of product that is continuously moving on a production line. A popular application in this area is Infeed machines, where the requirement is to put irregularly spaced product into pockets on a conveyor, so that they may be fed into a downstream machine.

The creation of 'order from disorder' is a taxing application of the controller, but has a large application base in the food, domestic product and pharmaceutical industry.

The following explains a simple Infeed system and the use of the **OFFSET** keyword.

The Infeed machine accepts irregularly spaced products on an input conveyor and feeds the product accurately into flights (pockets on a conveyor) on a *Parent machine*, typically a packaging machine that inserts product into boxes.

Product is fed onto the *Stripping/Synchronizing Conveyor*, which runs slightly faster than the Input conveyor to separate product. The *Product* is next transferred to the *Synchronizing Conveyor*, which runs at the same base speed as the *Parent machine*, determined by following a pulse train generated by an Encoder (the Parent Machine Encoder) which is connected to the auxiliary encoder input on the NextMove PC. This Encoder is geared to the *Parent machine* such that one turn (1000 pulses) corresponds to one flight spacing. The encoder is wrapped at the value of 1000 using the keyword **AUXENCODERWRAP**.

When a product passes the *Photocell* a fast interrupt is generated. The controller latches all encoder and positional data. The value of the **AUXENCODER** can then be used to determine the required offset to place the product accurately into a flight.

The correction is expressed as a positional error in the product with respect to the flight position. The MINT **OFFSET** command is used to perform the correction. **OFFSET** works much like a normal relative positional move, except that the acceleration-deceleration profile is imposed on the base speed of the *Parent machine*. The move profile of the offset move is controlled by the **OFFSETMODE** keyword. Offset mode 2 is a triangular move profile for smooth correction over a distance specified by **OFFSETDISTANCE**.

In practice it is often possible to have two products on the synchronizing conveyor at one time, therefore the program has been written so that it can simultaneously sense and correct product position. The program also ensures that the second correction is not implemented until the first package is fed into the *Parent machine*.

A further function of the controller is to provide machine status information to the operator, such as number of products per hour etc. This information can be displayed via any output channel.

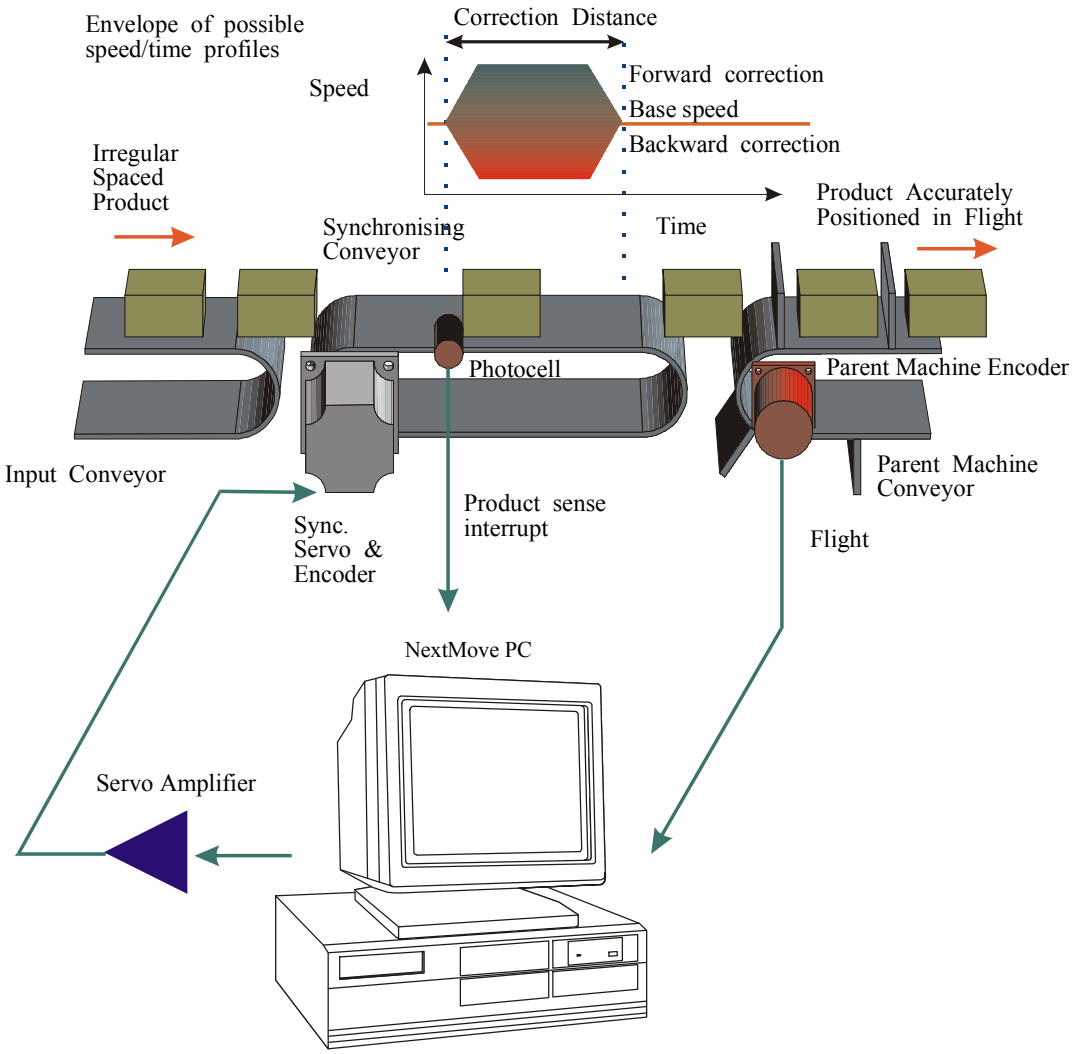


Figure 31: Diagram of Product Infeed Machine

The program is a simple example of the controller in an Infeed machine application. In more complicated applications, it is possible to collate a number of products and feed them into a single flight. The flexibility of the MINT programming language is such that these changes can be made quickly. The control program is stored in non-volatile RAM in the controller. As far as the operator is concerned, the system is a dedicated Automatic Product Infeed machine.

Infeed.MNT

File is in the MINT\NMPC\EXAMPLES directory.

Hardware Guide

7

This chapter describes in detail the hardware interface to the NextMove PC controller.

- ◊ Tour of the NextMove PC board
- ◊ Detailed description of the hardware interfaces
- ◊ State of the controller during system reset

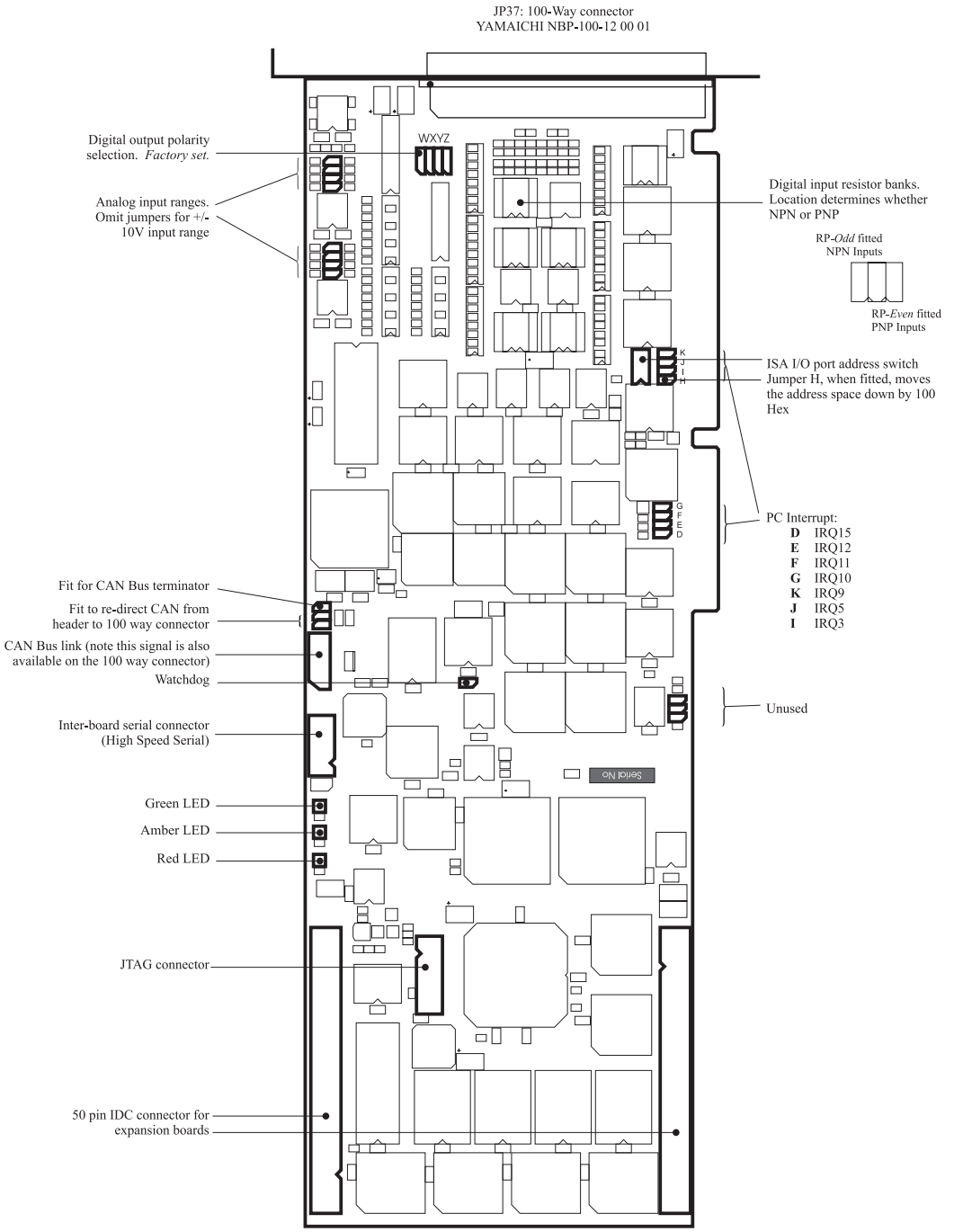


Figure 32: NextMove PC

7.1 Operating Environment

The safe operation of this equipment depends upon its use in the appropriate environment:

- At an altitude of ≤ 2000m (6560ft) above sea level
- In a locally ambient temperature of 0°C to 40°C (32°F to 104°F)
- In relative humidity levels of 80% for temperatures up to 31°C (87°F) decreasingly linearly to 50% relative humidity at 40°C (104°F), non-condensing
- The pollution degree according to IEC664 shall not exceed 2
- Power is supplied to the board via the PC power supply bus
- The atmosphere shall not contain flammable gases or vapours
- There shall not be abnormal levels of nuclear radiation or X-rays

7.2 Connection to the PCB

All of the standard interfaces are brought out on the 100 way D-type connector mounted on the metal bracket. The PCB connector is a YAMAICHI NBS-100-12 00 01 which will mate a YAMAICHI NBP-100-100100BF, a miniature ribbon cable connector. For the convenience of the user a kit consisting of a cable assembly and a DIN rail mounting break-out board is available. This is described in Section 8.

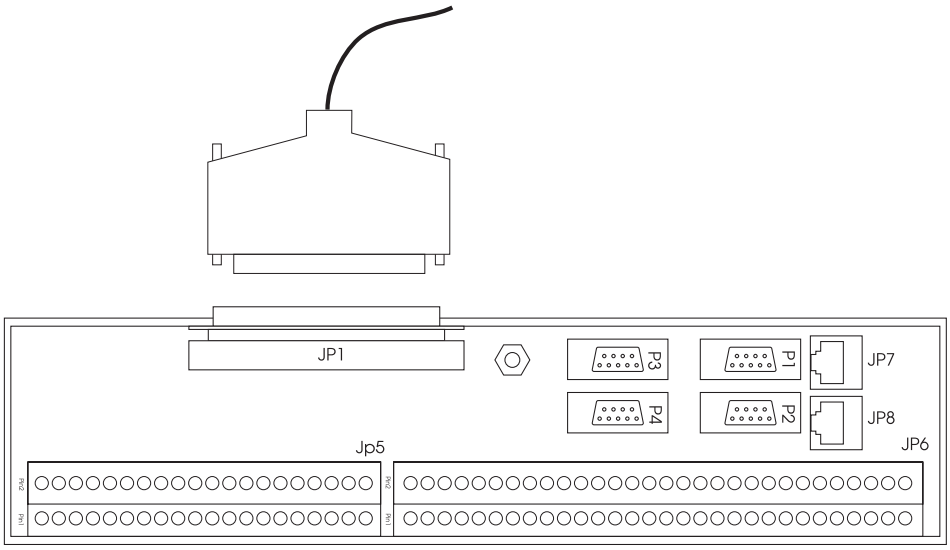


Figure 33: NextMove PC Breakout

The connections on the D connector are as shown below, shaded groups of pins are referred to a common potential.

Pin	Upper Row	Lower Row
1	ANALOG_GND	ADC_0
2	ADC_1	ADC_2
3	ADC_3	ADC_4
4	ADC_5	ADC_6
5	ADC_7	DAC_0
6	DAC_1	DAC_2
7	DAC_3	ANALOG_GND
8	DIG_OUT_0	DIG_OUT_1
9	DIG_OUT_2	DIG_OUT_3
10	DIG_OUT_4	DIG_OUT_5
11	DIG_OUT_6	DIG_OUT_7
12	DIG_OUT_8	DIG_OUT_9
13	DIG_OUT_10	DIG_OUT_11
14	USR-V+	USR_GND
15	CAN_WIRE+	CAN_WIRE-
16	!EXT.INT.U	ENC0.I
17	ENC0.!I	ENC0.B
18	ENC0.!B	ENC0.A
19	ENC0.!A	ENC1.I
20	ENC1.!I	ENC1.B
21	ENC1.!B	ENC1.A
22	ENC1.!A	ENC2.I
23	ENC2.!I	ENC2.B
24	ENC2.!B	ENC2.A
25	ENC2.!A	ENC3.I
26	ENC3.!I	ENC3.B
27	ENC3.!B	ENC3.A
28	ENC3.!A	USR-V+
29	PULSE.0	DIR.0
30	BOOST.0	PULSE.1
31	DIR.1	BOOST.0
32	PULSE.2	DIR.2
33	BOOST.2	PULSE.3
34	DIR.3	BOOST.3
35	USR_GND	DIG_IN0
36	DIG_IN1	DIG_IN2
37	DIG_IN3	DIG_IN4
38	DIG_IN5	DIG_IN6
39	DIG_IN7	DIG_IN8
40	DIG_IN9	DIG_IN10
41	DIG_IN11	DIG_IN12
42	DIG_IN13	DIG_IN14
43	DIG_IN15	DIG_IN16
44	DIG_IN17	DIG_IN18
45	DIG_IN19	DIG_IN20
46	DIG_IN21	DIG_IN22
47	DIG_IN23	GND
48	+5V	GND
49	+5V	RELAY_COM
50	RELAY_NC	RELAY_NO

Notes:

1. ADC_1 means input to the analog to digital converter channel one.
2. DAC_1 means output from digital to analog converter channel one.
3. DIG_IN1 means digital input one (w.r.t. USR-V+ and USR_GND).
4. DIG_OUT1 means digital output one (w.r.t. USR-V+ and USR_GND).
5. Each stepper motor axis has 3 control lines PULSE, DIR and BOOST, which are NPN open collector. The outputs are referred to the host's 0V (labelled GND).
6. !EXT.INT.U is a high speed interrupt line referenced to the host's 0V (labelled GND).
7. CAN_WIRE+ and CAN_WIRE-are the CAN connections. They are referred to the host's 0V (labelled GND).
8. The relay pins are fully floating.

7.3 Digital I/O

There are a total of 24 general purpose digital inputs and 12 general purpose digital outputs. The digital inputs are software configurable for any one of the following functions:

- **forward limit** (end of travel) input on any of the 8 axes.
- **reverse limit** input on any of the 8 axes.
- **home input** on any of the 8 axes.
- **drive error** input on any of the 8 axes.
- **stop input** (controlled) on any of the 8 axes.

The inputs can be programmed such that any of the axes can share the same input if necessary.

The inputs are also programmable in software for edge triggered (positive and negative) and their active level.

The digital outputs can be programmed as a **drive enable** output for any of the 8 axes. Again, axes can share the same output. The active level of the output is also software programmable.

As well as the general purpose I/O, *NextMove PC* also supports a fast position latch input (described in section 7.3.2) and optionally, digital outputs for stepper control. The stepper outputs can be programmed in software as general purpose outputs. The stepper outputs are discussed in section 7.5.

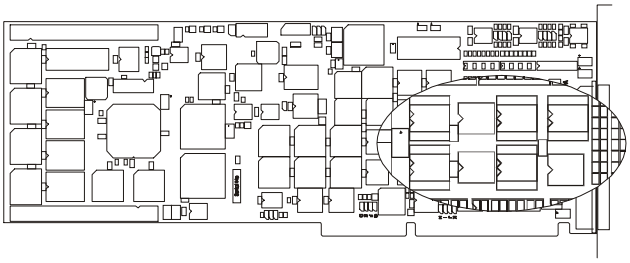
7.3.1 Digital Inputs

There are twenty-four optically isolated digital inputs.

The digital inputs use ac opto-isolators and may be referred to **usr-V+** for use with NPN drive transistors or low-side switches. Alternatively they may be referred **usr-gnd** for use with PNP drive transistors or high-side switches.

In either case the user provides an external supply which drives the inputs with a voltage in the range 5V to 24V relative to **usr-gnd** (this does not have to be **usr-V+)** or -5V to -24V relative to **usr-V+** (this does not have to be **usr-gnd**).

These are configured *at the factory* by fitting or omitting resistor banks as follows:



Inputs	NPN		PNP	
	Fit	Omit	Fit	Omit
0..3	RP1	RP2	RP2	RP1
4..7	RP3	RP4	RP4	RP3
8..11	RP5	RP6	RP6	RP5
12..15	RP7	RP8	RP8	RP7
16..19	RP9	RP10	RP10	RP9
20..23	RP11	RP12	RP12	RP11

The factory default is PNP for all inputs.

The input circuit is as follows:

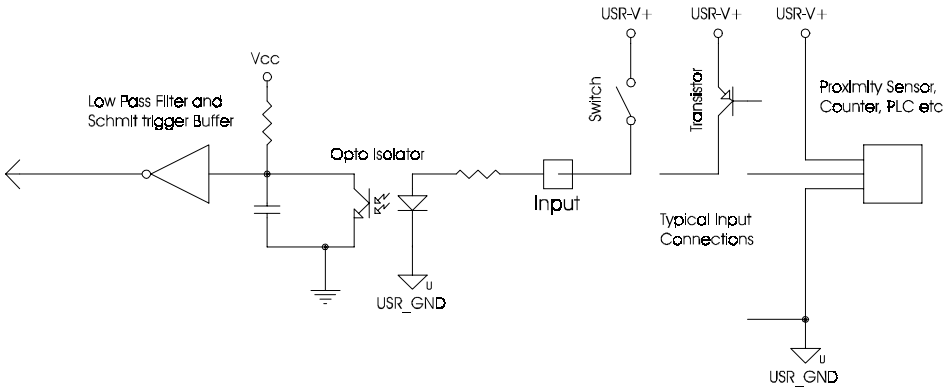


Figure 34: Digital Inputs, PNP configuration

The input is suitable to be driven by an (open collector) PNP transistor. The common for the inputs is connected to the negative of the user's power supply (**usr-gnd**). Any one of the inputs (**dig_in_1**) is connected via, (for the case of this example), a micro-switch to the positive of the user's power supply (**usr_V+**).

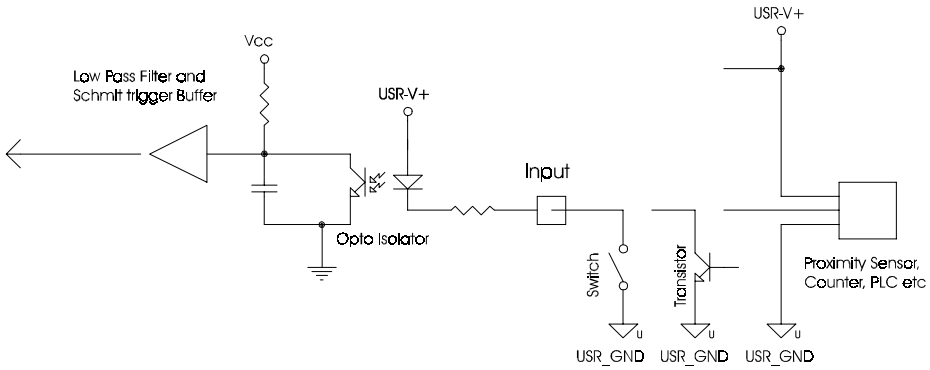


Figure 35:Digital Inputs, NPN configuration

The input is suitable to be driven by an (open collector) NPN transistor. The common for the inputs is connected to the positive of the user's power supply (**usr_V+**). Any one of the inputs (e.g. **dig_in_2**) is connected via, (for the case of this example), a micro-switch to the negative of the user's power supply (**usr_gnd**).

Note: Sustained voltages above 24V will damage the inputs.

The inputs are conditioned using low pass RC filters and *Schmitt* trigger buffers. An input pulse must have a duration of at least 2ms (one software scan) to guarantee acceptance by the application program when configured as edge triggered.

Associated MINT keywords are:

#INx, ACTIVEINLEVEL, IN, INx, INACTIVEMODE, NEGEDGEIN, POSEDGEIN

7.3.2 Fast Interrupt

The **fast interrupt** is a high speed processor interrupt which results in the latching of all the axis positions within 30µs. It can therefore be used in high speed print registration applications. When triggered, the position is captured and a user interrupt can be triggered to handle this event.

The user must drive the input **!EXT.INT.U** with a negative going pulse falling to 0.8V max. The signal is filtered and digitally de-coupled to give a single interrupt length pulse on the falling edge of the input. The input is *Schmitt* triggered to allow slow falling signals to function correctly. The rising edge does not generate an interrupt.

Mechanical switches are not recommended for this input as switch bounce may cause multiple interrupts. Noise on this input can also cause multiple interrupts as the input will react to signals only 100-200ns apart.

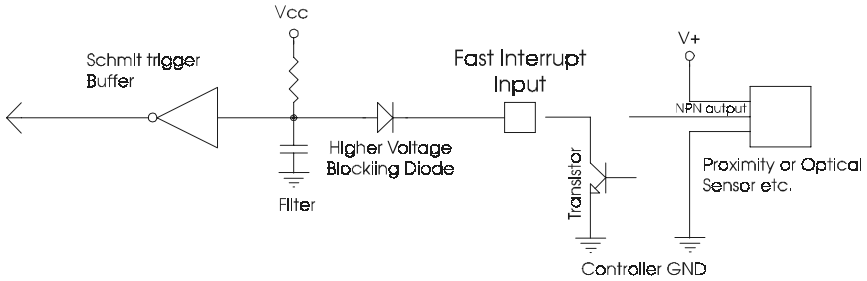


Figure 36: Fast Interrupt Circuit

Note: this signal is referred to the PC host, there is no isolation.

Associated MINT keywords are:

#FASTPOS, FASTPOS, FASTENCODER, FASTAUXENCODER,

7.3.3 Digital Outputs

There are twelve opto-isolated digital outputs. They are open collector type and are arranged as a bank of eight and a bank of four.

7.3.3.1 Output Polarity Selection

Each output bank may be configured *at the factory* to be either positive common (NPN) or negative common (PNP). The outputs use arrays of Darlington transistors as shown:

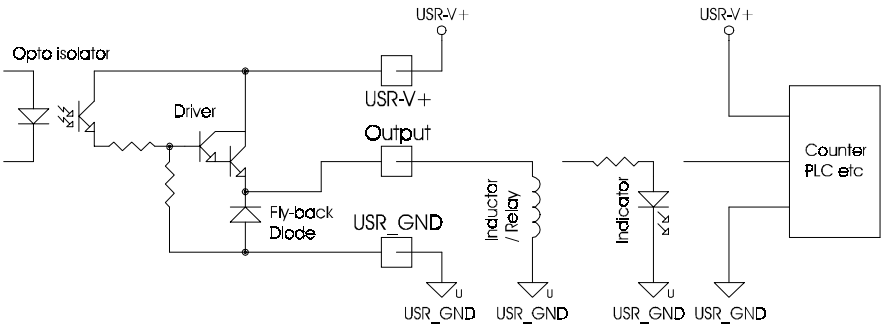


Figure 37: Digital Outputs PNP configuration

This refers to the type of transistor used to drive the output. In this case, PNP transistors. The common for the outputs is connected to the positive of the user's power supply (**usr_V+**). The load is connected between an output (e.g. **dig_out_1**) and the negative of the user's supply (**usr_gnd**).

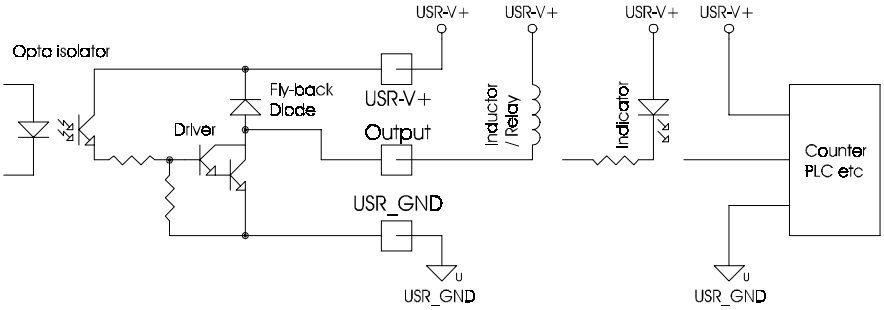


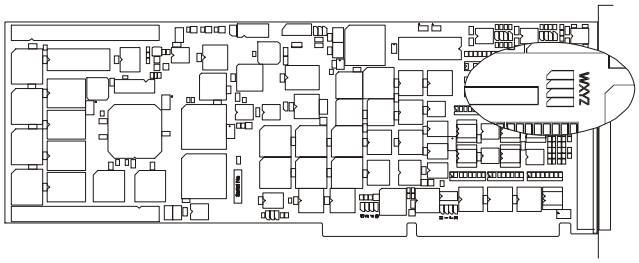
Figure 38: Digital Outputs NPN configuration

These outputs are driven by NPN transistors. The common for the outputs is connected to the negative of the user's power supply (**usr_gnd**). The load is connected between an output (e.g. **dig_out_1**) and the positive of the user's supply (usr_V+).

In each case there is a freewheeling/fly-back diode in parallel with the load, this allows inductive loads (such as relays) to be driven without the need for external diodes.

The outputs can each switch 50mA through a voltage of up to the applied user voltage. The outputs can switch higher currents but this has power dissipation limits. No single output may carry more than 350mA and no bank of outputs may carry more than 750mA total current. Exceeding these limits will cause damage to the device, a short circuit may even cause the driver to explode.

The selection of output polarity is made *at the factory* by a combination of jumper positions and driver type as follows:



Outputs	NPN (common is USR-V+)		PNP (common is USR_GND)	
	Device	**Jumper**	**Device**	**Jumper**
0..7	U37 = '2803	W and X to the right	U37 = '2982	W and X to the left
8..11	U38 = '2803	Y and Z to the right	U38 = '2982	Y and Z to the left

The factory default is PNP for all outputs.

The digital outputs are inactive following system **reset**.

7.3.3.2 NPN Outputs with Higher Voltages

The output drivers contain built-in freewheeling diodes which clamp the output to **usr-gnd** (on a PNP output) or **usr-V+** (on an NPN output). However, there may be users who wish to operate an NPN output as an open collector output where the load is connected to a potential greater than **usr-V+** this is possible but the user must first do the following:

1. Remove jumper **X** or **Z** altogether so that the built-in freewheeling diodes of that bank are disabled.
2. Connect a freewheeling diode across each inductive load connected to the bank whose freewheeling diodes have been disabled.
3. Ensure that the maximum voltage the devices is exposed to is less than 50V.

Damage may occur unless all of these steps are adhered to.

Associated MINT keywords are:

ACTIVEOUTLEVEL, OUT, OUTx

7.4 Analog I/O

7.4.1 Analog Inputs

The analog inputs are led to a MAX180: an eight channel, twelve bit, multiplexed analog-to digital converter. This ADC is capable of operating with either bipolar or unipolar input voltages, the selection is made under software control.

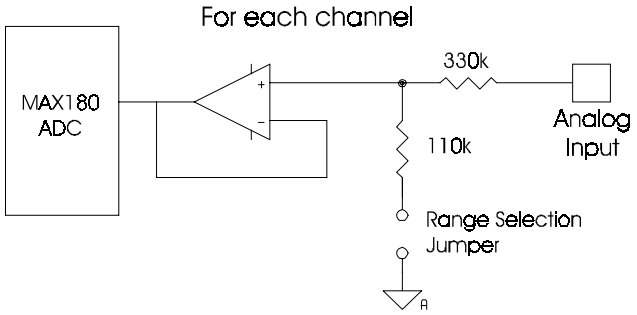


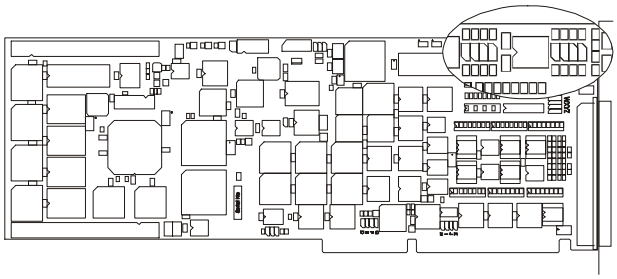
Figure 39: Analog Buffer and Range Selection

7.4.1.1 Input Range Selection

The input voltage can be in one of two ranges determined by fitting a jumper. If the jumper is fitted then the usable input range is ±10V only (the software must operate the ADC in bipolar mode). With this range the input impedance is 440k.

If the jumper is omitted then the input range of the ADC is software selectable to be either 0-5V or ±2.5V. With this range the input impedance is that of the input op-amp (of the order of mega ohms).

Each channel has an independent range selection jumper as detailed below:



Channel	Jumper	Channel	Jumper
0	R	4	N
1	S	5	O
2	T	6	P
3	U	7	Q

These jumpers are all fitted by default. If a channel is disconnected then it is advisable to fit the jumper as this presents a defined potential to the buffer circuit.

The analog inputs are sampled by MINT at a rate of 500Hz.

7.4.1.2 Differential Mode

The analog inputs can be read as 4 differential pairs. Reading channel 0 in differential mode gives channel 0 minus channel 1; reading channel 1 in differential mode gives channel 1 minus channel 0. This mode is not 'true differential' but can be used for measuring a signal relative to a reference rather than the controller ground.

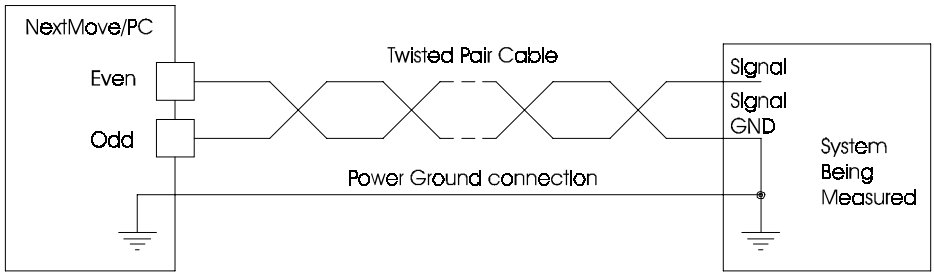


Figure 40: Analog Input, Differential Connection

Associated MINT keywords are:

ADCMODE, ANALOGUEx

7.4.2 Analog Outputs (Drive Demand/Command)

There are four 12 bit resolution analog outputs implemented in an MP7611. The outputs have a range of ±10V (4.9mV per bit). System **reset** causes the outputs to go to 0V. Outputs **0** to **3** are buffered by op-amps suitable for driving a load resistance of ≥10kΩ. The outputs are referred to PC system ground (not opto-isolated).

By default MINT™ and the MINT™ Motion Library use the analog outputs to control servo drives. Outputs **0** to **3** correspond to axes **0** to **3** respectively.

The analog output buffer circuit is shown below.

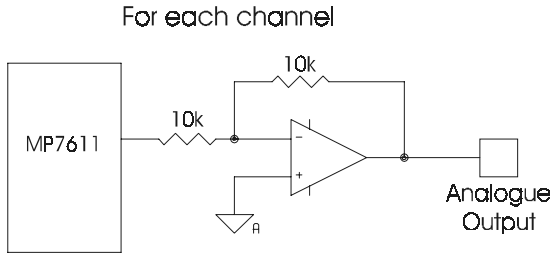


Figure 41: Analog Output buffer

Associated MINT keywords are:

CURRLIMIT, DAC, DACMODE, DEMAND

7.5 Encoder Interface

Up to four incremental encoders may be connected to the *NextMove PC* card. These encoders may provide either single-ended open collector; single-ended TTL or complementary TTL signals. The input circuit uses AM26LS32 differential line receivers with a biasing network so that either type of signal will be received correctly. Each encoder channel has inputs A, B and Index. The encoder interface supplies 5V to power the encoder electronics.

Note that the encoder power is drawn from is the PC 5V rail. The encoder inputs are referred to the PC GND.

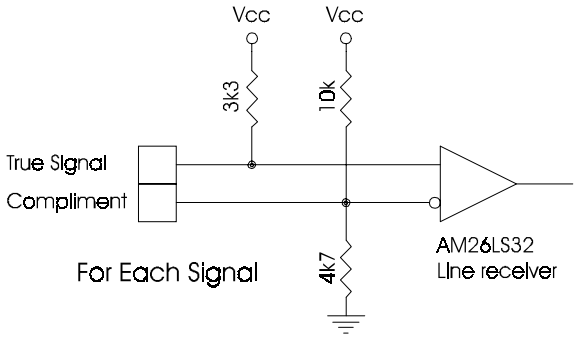


Figure 42: Encoder line receiver interface

The maximum input frequency the encoders can normally accept is 5.3 million quadrature counts per second. This equates to a maximum frequency for the A and B signals of 1.3MHz. However this maximum is limited to short cables, the table below shows recommended length vs. frequency for differential signals, single-ended TTL signals should be kept below 15m and open collector signals are limited to 250kHz and 10m (33ft).

Frequency	Max. Length
1.3MHz	50m (164ft approx.)
500kHz	200m (656ft approx.)
250kHz	300m (984ft approx.)
100kHz	500m(1640ft approx.)
50kHz	1000m (3280ft approx.)

The encoders are connected via 9 pin 'D' connectors on the breakout board (refer to Section 8 for details).

Associated MINT keywords are:

ENCODER, ENCODERSCALE, ENCODERVEL, ENCODERWRAP

7.6 Error Relay

A single pole change-over relay is included on the card to provide a volt-free contact for system enabling. The relay is controlled by a latch which is cleared during reset of the *NextMove PC* card. Reset occurs on power-down, watchdog error or under control of the host PC. The relay is energised under software control.

Due to the track rating on the PCB, the relay is limited to a rating of 150mA at 24V DC. For this reason the relay should be used for voltage free signal switching and should not be used for power.

Associated MINT keywords are:

ENABLE

7.7 Stepper Drive Outputs

There are 12 outputs which are used to form the stepper motor control outputs. The outputs may be programmed in software for the following functions:

- Step and direction for driving stepper motor drives.
- PWM and direction.
- Digital outputs for general purpose use.

Please refer to the *MINT for NextMove Programmers Manual* for a full explanation as to how these outputs can be used.

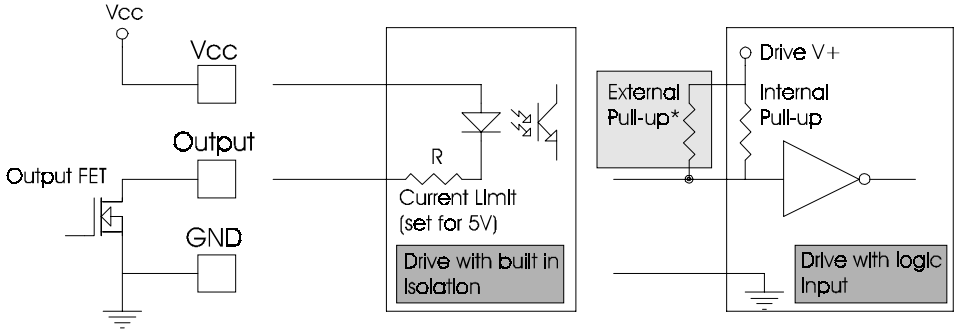
The outputs are all N-channel *mosfets* rated at 100mA@24V and are referred to system ground. The 12 outputs are arranged as four groups of 3 signals whose names and functions are shown in the following table:

Signal Name	Usage: Stepper Motor
pulse.n	step pulses
dir.n	direction of rotation
boost.n	general purpose control line

By default the software arranges the stepper motors to be axes four to seven. Therefore signals **pulse.0**, **dir.0** and **boost.0** in fact refer to axis four.

The outputs are all active low open collector signals and as such will need pull-up resistors when connecting to stepper drives without internal pull-ups.

When connecting to stepper drives with opto isolated inputs the outputs should be connected to the drive input negative or cathode connection, with the positive or anode connection connected to the voltage recommended in the drive instructions.



* External Pull up required only if there is no internal one

Figure 43: Stepper Drive Connections

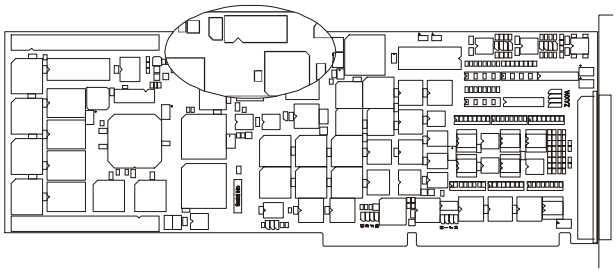
Associated MINT keywords are:

BOOST, PULSE, STEPDIRECTION

7.8 Communications

7.8.1 Serial Port

The processor has a proprietary on-chip serial port intended for high speed inter-processor communications at up to 1MBit/s. Access to it is made via a ten way, two row 2.54mm pitch header labelled *JP36*. The connections to this header are as follows:

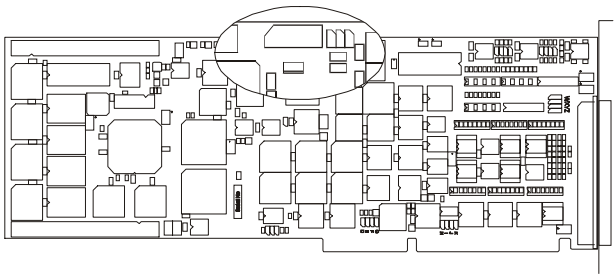


Pin	Signal
1	**+5V**
2	**0V**
3	**fsr**
4	**clkx**
5	**dr**
6	**dx**
7	**clkr**
8	**fsx**
9	**0V**
10	**+5V**

Note: The signals on this connector are referred to the PC host, there is no isolation and no buffering. The signals are arranged so that a cable wired to swap the pins (1 ⇔ 10, 2 ⇔ 9 etc.) will provide the correct interconnection of two *NextMove PC* cards.
The serial port is reserved for future use.

7.8.2 CAN Bus

CAN (Control Area Network) is a 1Mb/s local area network. A transceiver for the CAN is fitted to the *NextMove PC* card. Access to it is made via a 6 pin *Molex KK* series 2.5mm pitch header labelled *JP32*. The connections to this header are as follows:



Pin No.	Signal
1	**can1+**
2	**can1-**
3	**+5V**
4	**gnd**
5	**+12V**
6	**-12V**

Note: the signals on this connector are referenced to the PC host, there is no isolation.

This connector is designed to allow connection to a bulkhead connector or connector pair of the required type in the next AT slot plate. These can be RJ45 for connection to other Optimised Control CAN nodes or 9 pin 'D' connectors for connection to other networks such as *CANOpen*. The use of a crimp housing allows flexible pin-out arrangements.

If this type of connection is used, the two unmarked jumpers next to jumper M should be removed. This disconnects the breakout board from the network.

The CAN is also brought out to the break-out PCB on two RJ45 connectors (see Appendix A for details).

If the NextMove PC card is at the end of the network it is necessary to connect the termination resistor by fitting jumper M (this is the factory default).

A range of CAN based I/O expansion modules are available for *NextMove PC*:

Name	Order Code	Description
InputNode 8	ION001-503	8 opto-isolated digital inputs. DIN rail mount.
OutputNode 8	ION003-503	8 opto-isolated digital outputs. DIN rail mount.
RelayNode 8	ION002-503	8 relay outputs. DIN rail mount.
ioNode 24/24	ION004-503	24 opto-isolated digital inputs and 24 opto-isolated digital outputs. DIN rail mount.
KeypadNode	KPD002-502	28 key Operator panel with 20x4 character display
KeypadNode	KPD002-505	41 key Operator panel with 20x4 character display

Please contact a representative for details of these and any new devices which are available.

A very low error rate of CAN communication can only be achieved with a suitable wiring scheme. The following points should be observed:

1. CAN must be connected via twisted pair cabling. The connection arrangement is normally a simple multi-point drop. The CAN cables should have a characteristic impedance of 120Ω; and a delay of 5ns/m. Other characteristics depend upon the length of the cabling:

Cable length	Maximum bit rate	Specific resistance	Conductor area
0-40m (0-157ft)	1 Mbit/s	70mΩ	0.25-0.34mm^2
40m-300m (157ft-1180ft)	200 kbit/s	< 60mΩ	0.34-0.60 mm^2
300m-600m (1180-2362ft)	100 kbit/s	< 40mΩ	0.50-0.60 mm^2
600m-1000m (2362ft-3937)	50 kbit/s	< 26mΩ	0.75-0.80 mm^2

2. Terminators should be fitted at both ends of the network only.

3. To reduce RF emissions and more importantly, to provide immunity to conducted interference, shielded twisted pair cabling should be used. If two CAN channels are bundled in a cable then each requires a twisted pair.

Cable screens/shields should not be connected to 0V onconnector sk4 and sk5 as this will cause ground loop interference into the 0V plane on the processor board.

4. The 0V rails of all of the nodes on the network must be tied together through the CAN cabling. This ensures that the CAN signal levels transmitted by a *NextMove PC* or CAN peripheral devices are within the common mode range of the receiver circuitry of other nodes on the network.

Cables may also be purchased from the factory.

See section 10 for more details on using CAN peripherals.

Associated MINT keywords are:

CANBAUD, CANSTATUS, NODELIVE, NODETYPE, KEYPADNODE, READKEY, REMOTEBAUD, REMOTEDEBOUNCE, REMOTESETUP, REMOTEESTOP, REMOTEIN, REMOTEINPUTACTIVELEVEL, REMOTENODE, REMOTEOUT, REMOTEOUTPUTACTIVELEVEL, REMOTEOUTPUTERROR, REMOTERESET, REMOTESTASTUS, STATUSNODE

7.9 Reset State

During PC power-up the *NextMove PC* controller is held in **reset**. It will also go into **reset** if the 5V supply drops below approximately 4.75V in order to prevent any uncontrolled operation of any of the integrated circuits during power down.

When *NextMove PC* is in **reset** for any reason, most of the controlled interfaces fall into known states. This is explained further below.

7.9.1 Communications

At power up the CAN controller will be held in reset and will have no effect on the CAN bus. If a **reset** occurs during the transmission of a message CAN errors are likely to occur.

Dual port RAM will contain no information at power up but will be accessible by the PC. A **reset** during operation will cause the DPR to stay in its current state, that is, if it is locked it will remain so. The timer tick function, in location 6 of DPR, will stop. This is the best indication of a **reset** occurring.

7.9.2 Digital Outputs

All of the digital outputs are inactive on power up regardless of their polarity. They will return to the inactive state whenever **reset** occurs.

7.9.3 Analog Outputs

All analog outputs are set to 0V by hardware during power-up and will return to 0V on a **reset**. However the output buffer circuit may have a small offset and so the actual voltage seen at the output may not be zero.

7.9.4 Pulse Generator Outputs

The pulse generator chips cannot be reset into a known state by hardware and so the level of the outputs is indeterminate during power-up. They will, however, not produce a pulse train, and so will not drive stepper motors before the processor is powered and software has control. During software (controlled) reset the outputs are set to inactive.

A reset during operation will not reset the pulse generator chips. However it will disable the pulse train outputs so any motion on stepper systems will stop.

7.10 LEDs

There are three coloured LEDs (green, amber and red) on *NextMove PC* these are status indicators and can give valuable information on the state of the software running on *NextMove PC*. The LEDs can be in an *off*, *on* or *flashing* state. These states are depicted in the table by the following icons:

Green	Amber	Red	Description
Off	Off	Off	No power or abnormal state.
On	On	On	In **hardware reset**.
On	Off	Off	In software reset (usual state after a **host reset** command).
Flashing	Don't Care	Don't Care	Application running. Flashes at 0.5 Hz.
Flashing	Flashing	Don't Care	Software **system error** (generally a synchronous error). Flashes at 4Hz.
Flashing	Don't Care	Flashing	**Asynchronous error** (for example: limit switch tripped or following error). Flashes at 4Hz.
Flashing	Flashing	Flashing	All flashing **at the same rate**. Indicate that the boot loader has not located any RAM on the board.

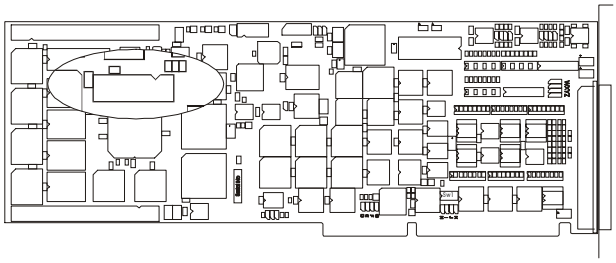
7.11 Expansion Board

Two box headers on the *NextMove PC* controller (as marked in Figure 49) provide an expansion bus for the controller. Two boards are directly supported by *NextMove PC*, both of which expand the axis capability of *NextMove PC*. These are discussed in section 10

The expansion board connections provide a direct interface to the 32 bit processor bus. Connection to expansion modules is through two 50-pin 0.1" pitch box headers. The expansion board is piggy-backed directly on the *NextMove PC*.

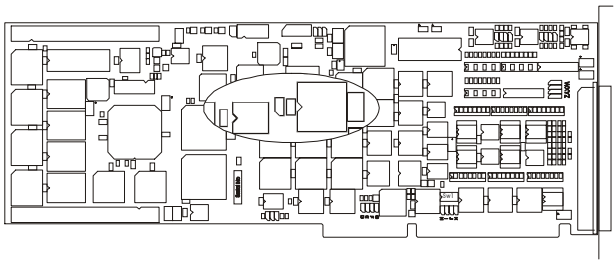
7.12 Miscellaneous

7.12.1 Emulator Connection



JP33 provides access to the processor for boundary scan emulation. The connections are those specified by *Texas Instruments*.

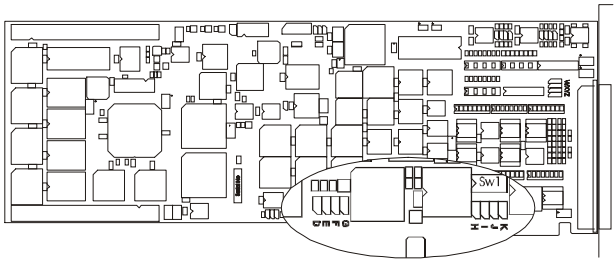
7.12.2 System Watchdog



Fitting jumper **L** enables the system watchdog (default). This causes the processor to reset in the event of a major firmware malfunction. It may be removed during low level code development and debugging.

7.12.3 Interrupt Level

The interrupt level is selected by fitting a jumper to **one** of several alternative positions :



Jumper Fitted	PC Interrupt	Typical Use
I	IRQ3	Serial Port 2
J	IRQ5	Parallel Port 2
K	IRQ9	Spare
G	IRQ10	Spare
F	IRQ11	Spare
E	IRQ12	Spare
D	IRQ15	Spare

It is possible to determine the use of interrupts 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.

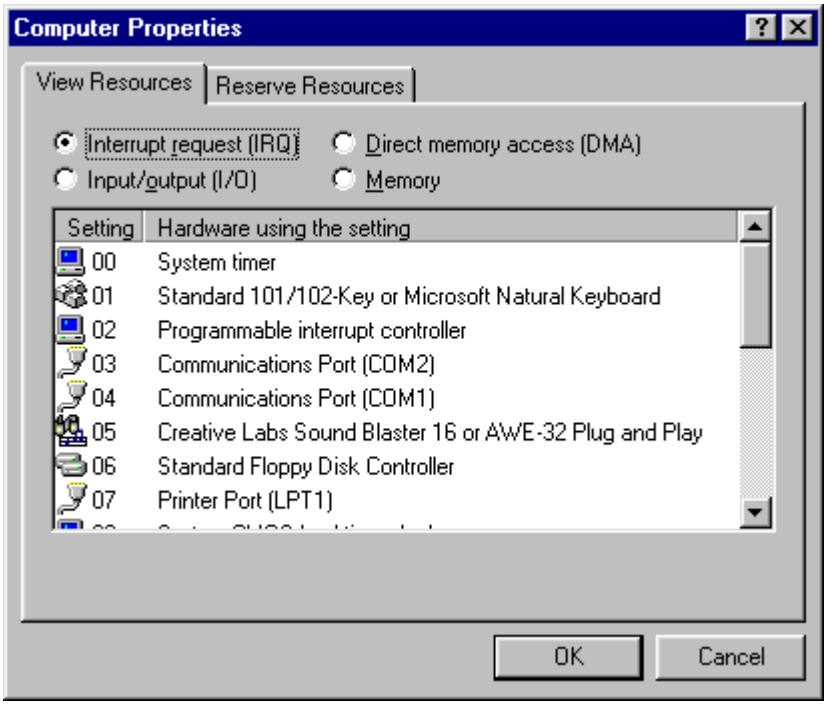


Figure 44: Windows '95 IRQ map

4. 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 *Interrupt request (IRQ)* is selected.

NextMove PC Breakout Board

8

The breakout board connects to the 100-pin connector on NextMove PC to provide convenient connectors for the I/O, CAN, encoders, servo command signals and steppers.

◊ Connection details.

The break-out module is approximately 292mm (11.50 inches) long by 70mm (2.76 inches) wide by 62 mm (2.45 inches) high. The layout of the board is shown in Figure 45. It fits on to either a 35mm symmetric DIN rail (EN 50 022, DIN 46277-3) or a G-profile rail (EN 50 035, DIN46277-1). The board 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.

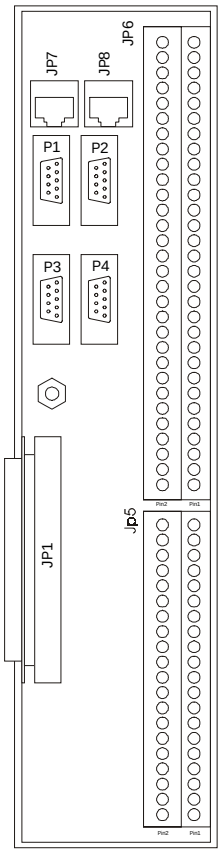


Figure 45: Breakout Board

The pinout of each connector and brief descriptions are given in the following sections.

8.1 Encoders: P1, P2, P3 and P4

The encoder inputs are brought out onto 9 pin D-type female sockets. The shell of the connector is connected to chassis screen/shield.

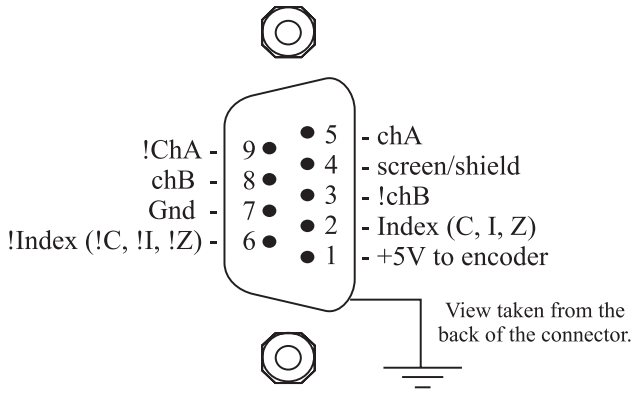
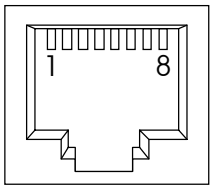


Figure 46: Encoder Pinout

Pin	Signal	Function
1	**+5V**	Power to the Encoder
2	**index**	Index true signal (channel I or Z or C)
3	**!chB**	Channel B compliment signal
4	**chassis**	Chassis connection
5	**chA**	Channel A true signal
6	**!index**	Index compliment signal (channel I or Z or C)
7	**gnd**	Power and signal ground
8	**chB**	Channel B true signal
9	**!chA**	Channel A compliment signal
Shell	**chassis**	360 degree cable screen/shield

8.2 CAN: JP7 and JP8

The CAN is presented on two shielded RJ-45 sockets.



Signal	Function	RJ-45 Pin
can+	CAN data signal	1
can-	CAN data signal	2
can V+	Power to the CAN Nodes	5
gnd	Chassis connection	4
chassis	Screen pin	-
NC	No Connection	3,6,7,8
chassis	360 degree cable screen	Shell

8.3 Terminal Block JP5

Pin	Signal	Function
1	**agnd**	Analog Ground
2	**anin0**	Analog Input 0 (0+)
3	**anin1**	Analog Input 1 (0-)
4	**anin2**	Analog Input 2 (2+)
5	**anin3**	Analog Input 3 (2-)
6	**agnd**	Analog Ground
7	**agnd**	Analog Ground
8	**anin4**	Analog Input 4 (4+)
9	**anin5**	Analog Input 5 (4-)
10	**anin6**	Analog Input 6 (6+)
11	**anin7**	Analog Input 7 (6-)
12	**agnd**	Analog Ground
13	**demand 0**	Demand Signal 0/Analog Output 0
14	**agnd**	Demand Reference/ Analog Ground
15	**demand 1**	Demand Signal 1/ Analog Output 1
16	**agnd**	Demand Reference/ Analog Ground
17	**demand 2**	Demand Signal 2/ Analog Output 2
18	**agnd**	Demand Reference/ Analog Ground
19	**demand 3**	Demand Signal 3/ Analog Output 3
20	**agnd**	Demand Reference/ Analog Ground
21	**usr-V+**	User power for digital I/O, 5 to 24V
22	**out0**	Digital Output bit 0
23	**out1**	Digital Output bit 1
24	**out2**	Digital Output bit 2
25	**out3**	Digital Output bit 3
26	**usr-gnd**	User ground for digital I/O
27	**usr-V+**	User power for digital I/O, 5 to 24V
28	**out4**	Digital Output bit 4
29	**out5**	Digital Output bit 5
30	**out6**	Digital Output bit 6
31	**out7**	Digital Output bit 7
32	**usr-gnd**	User ground for digital I/O
33	**usr-V+**	User power for digital I/O, 5 to 24V
34	**out8**	Digital Output bit 8
35	**out9**	Digital Output bit 9
36	**out10**	Digital Output bit 10
37	**out11**	Digital Output bit 11
38	**usr-gnd**	User ground for digital I/O
39	**fast-int**	Fast Interrupt Input (not isolated) **!EXT.INT.U**
40	**gnd**	Digital Ground

The shaded pins show connections referenced to USR-V+ and USR-GND. All other pins are at the potential of the host.

8.4 Terminal Block JP6

Pin	Signal	Function
1	**usr-V+**	User power for digital I/O, 5 to 24V
2	**in0**	Digital Input bit 0
3	**in1**	Digital Input bit 1
4	**in2**	Digital Input bit 2
5	**in3**	Digital Input bit 3
6	**usr-gnd**	User ground for digital I/O
7	**usr-V+**	User power for digital I/O, 5 to 24V
8	**in4**	Digital Input bit 4
9	**in5**	Digital Input bit 5
10	**in6**	Digital Input bit 6
11	**in7**	Digital Input bit 7
12	**usr-gnd**	User ground for digital I/O
13	**usr-V+**	User power for digital I/O, 5 to 24V
14	**in8**	Digital Input bit 8
15	**in9**	Digital Input bit 9
16	**in10**	Digital Input bit 10
17	**in11**	Digital Input bit 11
18	**usr-gnd**	User ground for digital I/O
19	**usr-V+**	User power for digital I/O, 5 to 24V
20	**in12**	Digital Input bit 12
21	**in13**	Digital Input bit 13
22	**in14**	Digital Input bit 14
23	**in15**	Digital Input bit 15
24	**usr-gnd**	User ground for digital I/O
25	**usr-V+**	User power for digital I/O, 5 to 24V
26	**in16**	Digital Input bit 12
27	**in17**	Digital Input bit 13
28	**in18**	Digital Input bit 14
29	**in19**	Digital Input bit 15
30	**usr-gnd**	User ground for digital I/O
31	**usr-V+**	User power for digital I/O, 5 to 24V
32	**in20**	Digital Input bit 12
33	**in21**	Digital Input bit 13
34	**in22**	Digital Input bit 14
35	**in23**	Digital Input bit 15
36	**usr-gnd**	User ground for digital I/O
37	**pulse0**	Pulse Output 0
38	**pulse1**	Pulse Output 1

Pin	Signal	Function
39	**dir0**	Direction Output 0
40	**dir1**	Direction Output 1
41	**boost0**	Boost Output 0
42	**boost1**	Boost Output 1
43	**gnd**	Digital Ground
44	**gnd**	Digital Ground
45	**pulse2**	Pulse Output 2
46	**pulse3**	Pulse Output 3
47	**dir2**	Direction Output 2
48	**dir3**	Direction Output 3
49	**boost2**	Boost Output 2
50	**boost3**	Boost Output 3
51	**gnd**	Digital Ground
52	**gnd**	Digital Ground
53	**relay-com**	Error Relay Common Connection
54	**relay-nc**	Error Relay Normally Closed Connection
55	**relay-no**	Error Relay Normally Open Connection
56	**chassis**	Chassis ground connection for cable shield
57	**gnd**	Digital Ground
58	**+5V**	
59	**usr-gnd**	User ground for digital I/O
60	**usr-V+**	User power for digital I/O, 5 to 24V

The shaded pins show connections referenced to USR-V+ and USR-GND. All other pins are at the potential of the host.

The relay pins are fully floating.

CE Marking

9

This chapter provides details on CE marking of NextMove PC

9.1 Directives

CE marking indicates that a product complies with all of the relevant directives. A brief discussion of the directives follows.

9.1.1 Machinery Directive 89/392/EEC

This relates to the safety of machinery. Any machine having a moving part has to comply with the essential requirements of this directive. This also applies to some mechanical sub-systems, such as fans. Electronic sub-systems, such as *NextMove PC*, do not fall within its scope. It should be noted that to meet the safety requirements of this directive, the primary machine safety system cannot be based on controller software. The paying of a notified body to audit the software and provide a safety case is prohibitively expensive. A hard-wired system using guard switches (to protect personnel or equipment) should be used.

9.1.2 Low Voltage Directive 72/23/EEC

Applies to equipment powered from 75V dc to 1500V dc or 50V ac to 1000V ac. Therefore this directive does not apply to *NextMove PC*.

9.1.3 EMC Directive 89/336/EEC

This directive applies to all 'apparatus liable to cause disturbance or the performance of which is liable to be affected by such disturbance'. The expression 'apparatus' is defined for the purpose of the regulations as '...an electrical or electronic appliance or system consisting of a finished product or products having an intrinsic function which is intended for the end user, and is supplied or intended for supply or taken into service or intended to be taken into service as a single commercial unit'.

This wording brings automated machines within the scope of the directive but not necessarily their electronic sub-systems. This is because of their lack of intrinsic function to the end user. However, as *NextMove PC* is built to issue E202-4A it will be CE marked with respect of the EMC directive, subject to the following conditions.

9.2 EMC Performance of *NextMove PC*

The data applies to products built to D841-4 or later. It should be noted that the equipment is suitable for light or heavy industrial use.

Test	Standard	Comments
EN55081-1 Generic emissions, light industrial	EN55022 Radiated emissions	Connected to a dedicated test board by 3m cable, Passed the B limit.
EN55082-2 Generic immunity, heavy industrial	IEC801-2 ESD	Contact to PC case, had no effect. Contact to shells of D-type connectors had no effect.
	IEC801-3 Radiated immunity	Connected to a dedicated test board by 3m cable. Test covered 10V/m 80-500MHz with no modulation. Also tested in a TEM cell, equipment functioned correctly throughout.
	IEC801-4 Fast burst transients	1kV via clamp to **main cable**, equipment functioned correctly throughout apart from Fast interrupt and analogue inputs, these self recovered.

9.3 Conditions of CE Marking

The apparatus shall be connected in accordance with the recommendations of this manual. There are certain restrictions regarding the length and type of cable:

Cable type	Ports
unrestricted	**usr-V+**, **usr-gnd**, user inputs, user outputs, **home**, **limit**, **drive error**, **power** input, **drive enable** relay
unrestricted but screening/shielding is advised	pulse, direction, reset, demand/command, stop, fast interrupt.
must be screened/shielded	
screened/shielded twisted pair	encoders, **CAN**
< 2.9m (10ft) long	Main cable to breakout board.

NextMove PC Axis Expansion Board

10

The standard form of NextMove PC allows the user to control up to four stepper axes and up to four servo axes. The Expansion Board allows the NextMove PC user to control up to eight stepper axes eight servo axes or any mixture of stepper and servo subject to a limit of eight axes in total.

- ◊ Overview and order codes.
- ◊ Connection and wiring information.
- ◊ Servo expansion breakout board.

The standard form of NextMove PC allows the user to control up to four stepper axes and up to four servo axes. The Expansion Board allows the NextMove PC user to control up to eight stepper axes eight servo axes or any mixture of stepper and servo subject to a limit of eight axes in total. The Expansion Board can also be used in a stepper system to provide encoder verification.

Features:

- 4 axes of servo control, signals brought out via a 37 pin D connector. Optional breakout board is available.
- 4 axes of stepper control, signals brought out via a 9 pin D connector.
- Master encoder interface, signals brought out via a 9 pin D connector.

Ordering Information:

Name	Order Code	Description
4 axes Servo/Stepper Expansion Board	OPT009-501	*NextMove PC* supports up to 8 axes of servo and stepper control. This expansion board allows up to 8 axes of servo or 8 axes of stepper or any axis mix in between. When used for stepper only, this board can be used to provide encoder verification for all 8 axes of stepper. The full expansion board also includes a master encoder input.
4 axes Stepper Expansion Board	OPT011-501	Provides a further 4 axes of stepper control giving a full 8 axes of stepper control
NextMove PC Expansion Breakout Board	OPT010-501	Breakout board for servo signals. For use with OPT009-501.
NextMove PC Expansion Cable - 37 way, 1 metre	CBL006-501	Cable for use with OPT010-501.
NextMove PC Expansion Cable - 37 way, 2 metre	CBL006-502	Cable for use with OPT010-501.

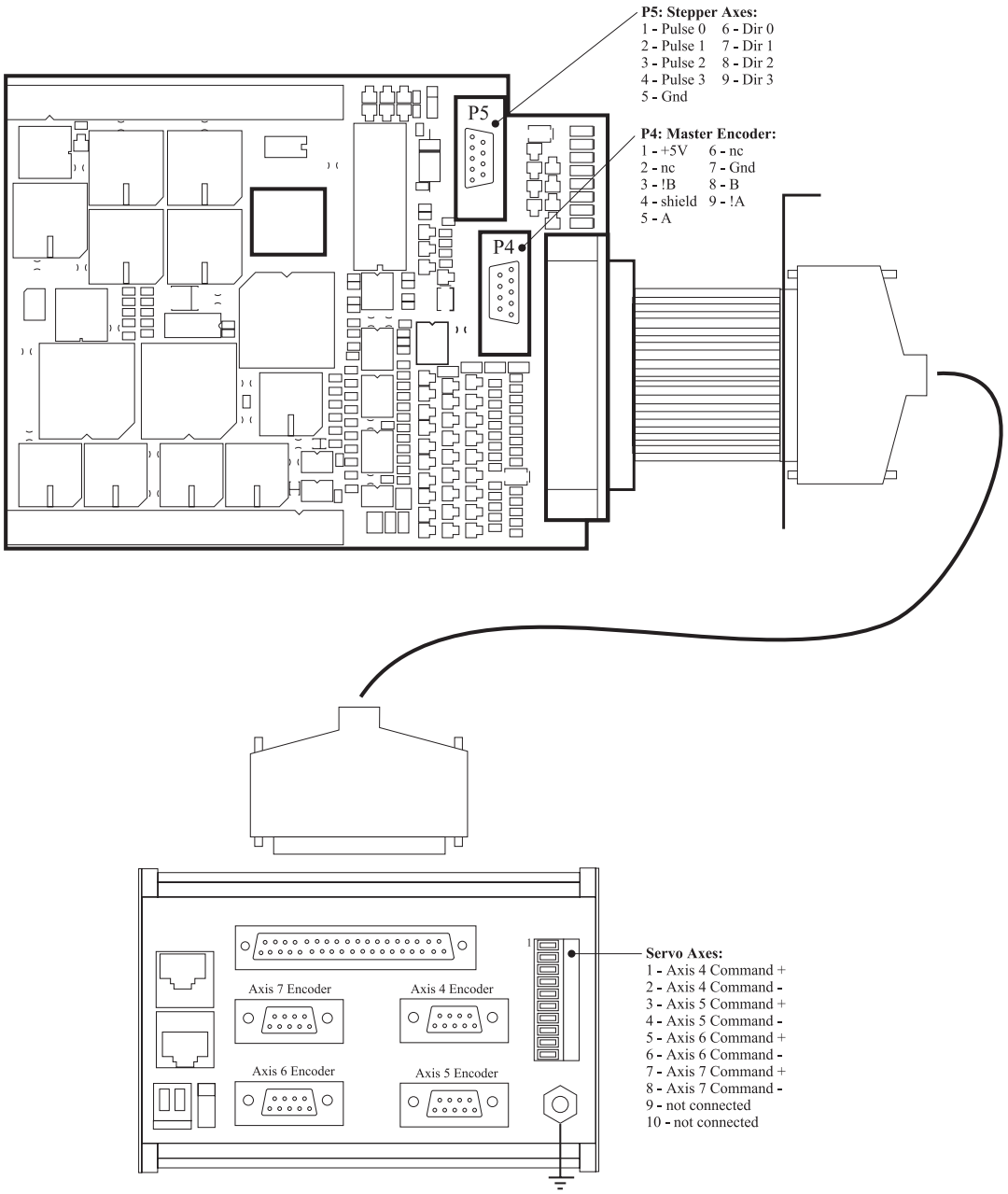


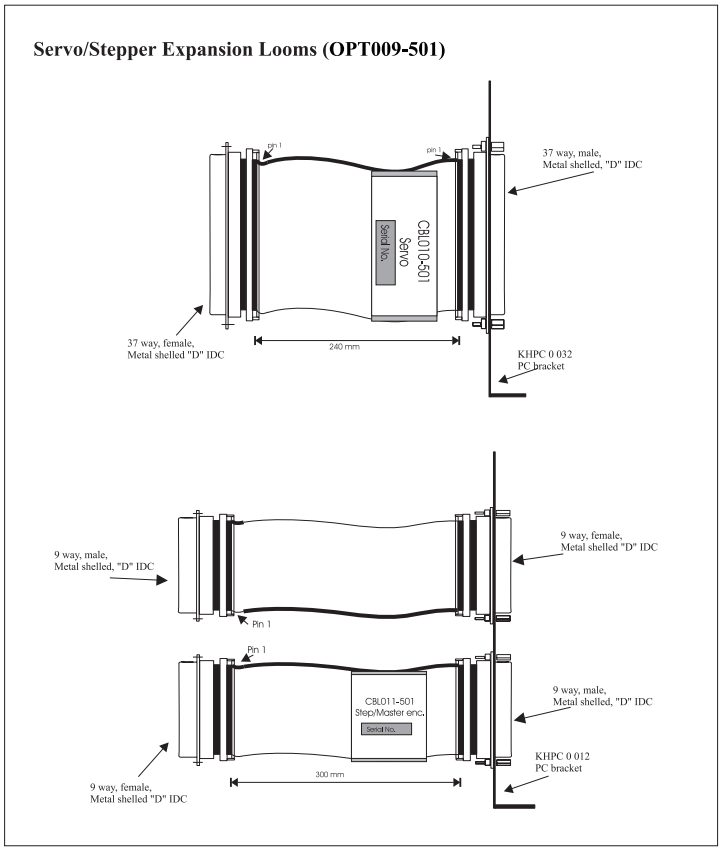
Figure 47: NextMove PC Axis Expansion Board with Servo Breakout Board

The area occupied by the board is shown as the shaded area in Figure 49. Connections to the motor drives are brought out via a separate slot on the PC. The PCB is 148mm long by 115mm deep. When fitted to NextMove PC it will project out and occupy space in the next slot. This means that the host PC will have to make two full-length, full height (4.8") slots available to accommodate the combination of NextMove PC and the expansion board.

The expansion board plugs directly on to a NextMove PC card via the 50 way connectors at the end farthest from the metal bracket. The servo connections are presented to a 37-pin D-type connector, the master encoder is brought out on a 9-pin D-type connector and the stepper connections are also brought out on a 9-pin D-type connector. To bring the signals out of the PC it is necessary to fit a metal bracket to the adjacent slot or slots in the PC and use the D-type to D-type IDC looms provided (see Figure 48).

The table below summarises the slot and bracket requirements:

Usage	**Full slots required**	**Brackets required**
More than four servo axes	2	one on NextMove, one 37-way for the servo
More than four servo axes + master encoder	2	one on NextMove, one 37-way for the servo, one 9-way for the master encoder
More than four stepper axes	2	one on NextMove, one 9-way for the stepper
More than four stepper axes + master encoder	2	one on NextMove, one dual 9-way for the stepper and master encoder



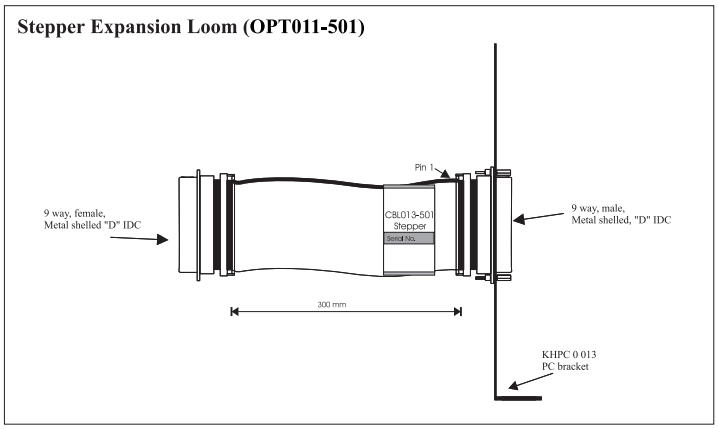


Figure 48: Expansion Looms

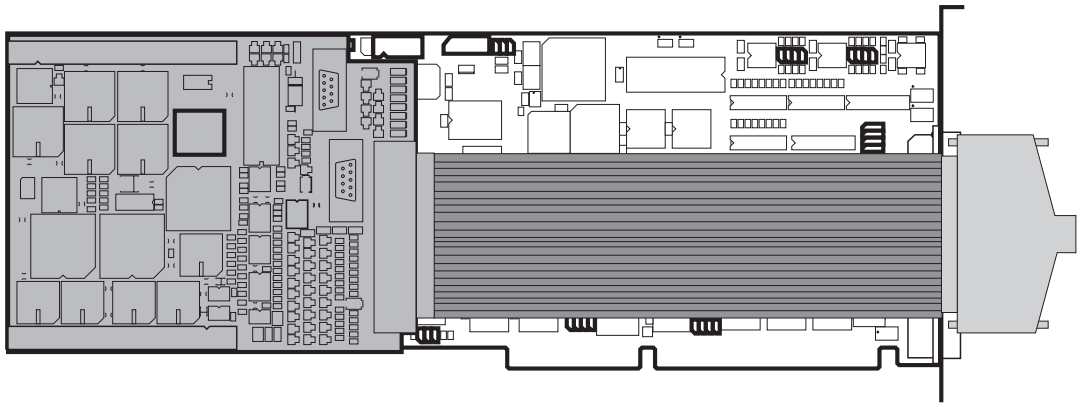


Figure 49: NextMove PC Expansion Board Connections

The expansion board is piggy-backed directly on the *NextMove PC* with the looms shown Figure 48 taking the signals from the connectors to the back of the PC.

The NextMove PC axes are designated 0-3 for the servo axes and 4-7 for the stepper axes. The expansion axes are numbered 0-3 for the stepper axes and 4-7 for the servo axes. This scheme allows axes of a given type to be numbered consecutively as required by the software.

When the expansion board is fitted, MINT can be used to determine whether NextMove is communicating with the board. From the MINT command line type:

```
VIEW HARDWARE
```

MINT should come back with the following:

Without Expansion Board

```
NextMove PC Configuration
 8 Axes of control
   4 Servo axes
   4 Stepper axes
 24 Digital inputs
 12 Digital outputs
 8 Analog inputs
 4 Analog outputs
 0 Auxiliary encoder inputs
 1 CAN channels.
```

With Expansion Board

```
NextMove PC Configuration
 8 Axes of control
   8 Servo axes
   8 Stepper axes
 24 Digital inputs
 12 Digital outputs
 8 Analog inputs
 4 Analog outputs
 1 Auxiliary encoder inputs
 1 CAN channels.
```

10.1 Connection Information

10.1.1 Servo Signals

The servo signals are brought out onto a 37 pin female D-type connector, which is in turn connected to a loom and standard PC bracket. The 37 pin connector can connect to the optional breakout board for ease of wiring. This is described in section 10.2.

Signal	Pin No.	Pin No.	Signal
GND	1		
GND	2	20	+5v
CAN2+	3	21	CAN2-
CAN1+	4	22	CAN1-
ENC7-B	5	23	ENC7-!B
ENC7-A	6	24	ENC7-!A
ENC7-I	7	25	ENC7-!I
ENC6-B	8	26	ENC6-!B
ENC6-A	9	27	ENC6-!A
ENC6-I	10	28	ENC6-!I
ENC5-B	11	29	ENC5-!B
ENC5-A	12	30	ENC5-!A
ENC5-I	13	31	ENC5-!I
ENC4-B	14	32	ENC4-!B
ENC4-A	15	33	ENC4-!A
ENC4-I	16	34	ENC4-!I
AGND	17	35	AGND
DEMAND6	18	36	DEMAND7
DEMAND4	19	37	DEMAND5

There are four 12 bit resolution analog demand outputs implemented in a DAC4815. The outputs have a range of ± 10V. System reset causes the outputs to go to 0V. The outputs are buffered by op-amps suitable for driving a load resistance of ≥10kΩ. The outputs are referred to PC system ground and are not isolated.

Refer to section 7.5 for details on the encoder interface.

10.1.2 Master Encoder

In addition to the four encoder channels for the servos, there is a Master encoder channel for encoder following. Refer to section 7.5 for details on the encoder interface. The connections are bought to the female 9 way D type connector as follows:

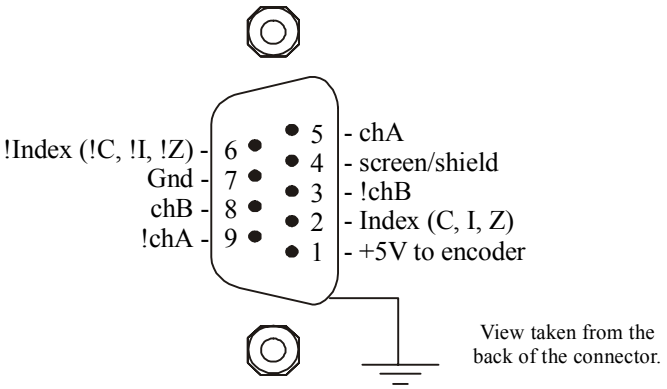


Figure 50: Encoder Pinout

Pin	Signal	Function
1	**+5V**	Power to the Encoder
2	**nc**	Not connected
3	**!chB**	Channel B compliment signal
4	**chassis**	Chassis connection
5	**chA**	Channel A true signal
6	**nc**	Not connected
7	**gnd**	Power and signal ground
8	**chB**	Channel B true signal
9	**!chA**	Channel A compliment signal
Shell	**chassis**	360 degree cable screen/shield

10.1.3 Stepper Outputs

Each stepper motor has two connections, PULSE and DIRECTION, which are open drain MOSFETs. The channels are numbered 0 to 3 with the connections bought to the male 9 pin D type connector as shown:

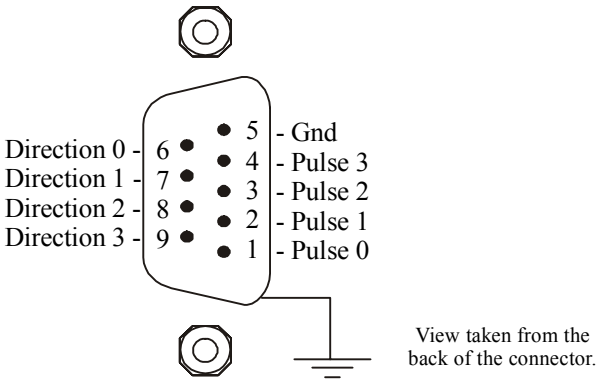


Figure 51: Stepper 9-pin pin-out

Pin No.	Signal
1	Pulse 0
2	Pulse 1
3	Pulse 2
4	Pulse 3
5	GND
6	Direction 0
7	Direction 1
8	Direction 2
9	Direction 3

10.2 Servo Breakout Board

The servo breakout board connect to the 37-pin connector on the axis expansion board to provide convenient connectors for the encoders and servo command signals.

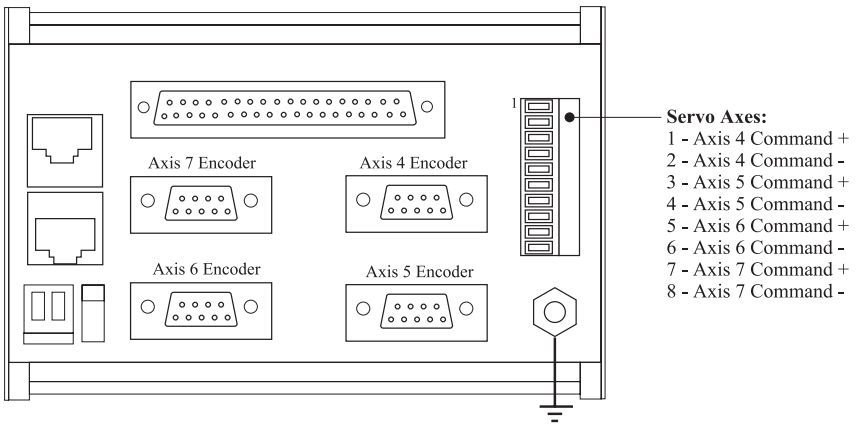


Figure 52: Servo Breakout Board

10.2.1 J3: Demand/Command Signals

The analog drive command signals are brought out onto a 3.5mm screw terminated connector.

Pin	Description
1	Axis 4 analog **command**+ signal
2	Axis 4 analog ground
3	Axis 5 analog **command**+ signal
4	Axis 5 analog ground
5	Axis 6 analog **command**+ signal
6	Axis 6 analog ground
7	Axis 7 analog **command**+ signal
8	Axis 7 analog ground
9	Not connected
10	Not connected

10.2.2 Encoders: P2, P3, P4 and P5

Refer to section 7.5 for details on the encoder interface.

The encoder inputs are brought out onto 9 pin D-type female sockets. The shell of the connector is connected to chassis screen/shield.

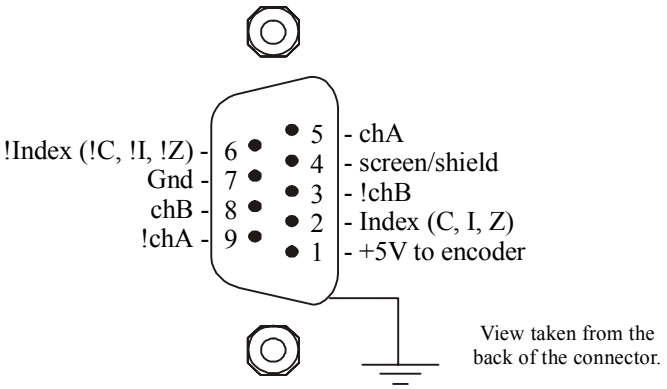


Figure 53: Encoder Pinout

Pin	Signal	Function
1	**+5V**	Power to the Encoder
2	**index**	Index true signal (channel I or Z or C)
3	**!chB**	Channel B compliment signal
4	**chassis**	Chassis connection
5	**chA**	Channel A true signal
6	**!index**	Index compliment signal (channel I or Z or C)
7	**gnd**	Power and signal ground
8	**chB**	Channel B true signal
9	**!chA**	Channel A compliment signal
Shell	**chassis**	360 degree cable screen/shield

Getting Started with CAN Peripherals

11

This chapter provides an introduction to the CAN peripherals and how these are configured to operate with NextMove PC. This chapter makes use of MINT to configure both the nodes and NextMove PC.

- ◊ Configuring a node
- ◊ Selecting CAN bus channels, node numbers and baud rates.
- ◊ MINT keyword summary for the CAN peripherals

A summary of the CAN Peripheral supported MINT keywords can be found in Section 4.2.

Before starting, it is important to make sure that *NextMove PC* is CAN Peripheral ready. This can be found by typing **VER** at the MINT command line:

```
C>VER
```

The MINT version number will be displayed:

```
MINT(tm) for NextMove PC.  Version 3.8/F.
```

The version number must be 3.8 or higher. To check that CAN channels are supported, type the following code at the MINT command line:

```
C>VIEW PLATFORM
```

MINT will return a configuration list of the from:

```
NextMove PC Configuration
 8 Axes of control
   4 Servo axes
   4 Stepper axes
 24 Digital inputs
 12 Digital outputs
 8 Analog inputs
 4 Analog outputs
 0 Auxiliary encoder inputs
 1 CAN channels.
```

The number of CAN channels supported will be displayed at the bottom of the list.

11.1 Network Possibilities

NextMove PC supports the full range of CAN Peripherals (*InputNode 8, OutputNode 8, RelayNode 8, IoNode 24/24* and *KeypadNode)*. Any number of CAN Peripherals, up to the maximum of 63, in any combination, are supported on the network. The restriction is that only four *KeypadNodes* are supported.

The following table shows the nodes that are supported and their *signatures* (type number, MINT and 'C' constants) which are used when adding the nodes to the network. Refer to Section 11.2 for implementation.

Type	MINT Constant	'C' Constant	Node type
0	`_none`	`ntNONE`	Not present
1	`_inputnode8`	`ntINPUT_NODE_8`	8 digital input node
2	`_outputnode8`	`nt_OUTPUT_NODE_8`	8 digital output node
3	`_relaynode8`	`ntRELAY_NODE_8`	8 relay output node
4..7			*Reserved*
8	`_ionode24_24`	`ntIONODE24_24`	24/24 IO node
9	`_keypad`	`ntKEYPAD`	*KeypadNode*

11.2 Quick Start

It is possible to get configured in a matter of minutes. All CAN Peripherals have a default configuration that is compatible with *NextMove PC*:

CAN Peripheral Type	Default Values	
	CAN baud rate	Node ID
InputNode 8	125	1
OutputNode 8	125	7
RelayNode 8	125	7
KeypadNode	125	14
ioNode 24/24	125	8

NOTE: If the CAN Peripheral's configuration has been changed, it will need to be statically reconfigured (refer to Section 11.3.5).

NextMove PC supports CAN Peripherals on its CAN Bus channel 1. All that is required is to power up *NextMove* and CAN Peripheral, connect their CAN ports (CAN1 on the controller), load MINT using *NextMove* WorkBench and add the CAN Peripheral to the network. It is now possible to start controlling remote I/O.

The following exercise, using *NextMove PC* as the host controller, illustrates the ease of configuration. This example uses a *RelayNode 8* CAN Peripheral. Refer to the configuration diagram represented in Figure 54.

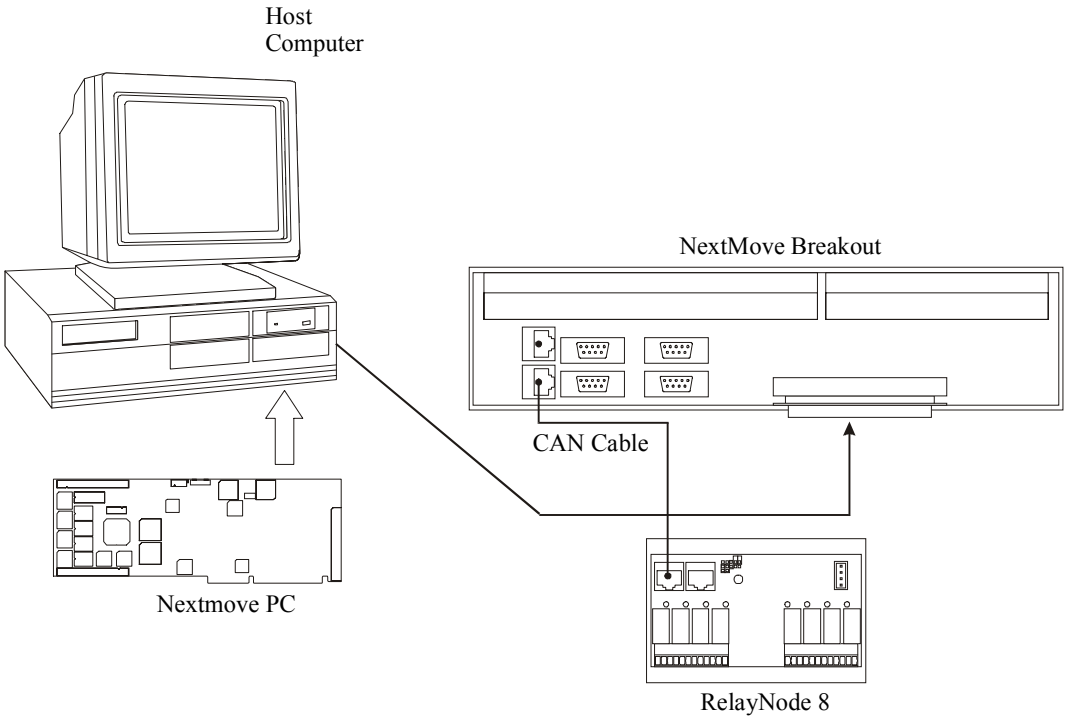


Figure 54: *NextMove PC* **connected to one *RelayNode 8*.**

11.2.1 Jumper setings

1. Make sure that jumpers JP1 and JP2 on the *RelayNode8* are fitted in position 1 (CAN1), as shown in Figure 55.
2. Make sure that jumper JP3 on the *RelayNode8* is fitted, as shown in Figure 55.
3. Make sure that jumpers JP4 and JP5 on the *RelayNode* are not fitted. Refer to Figure 55.

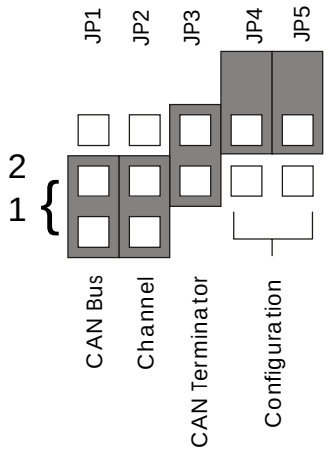


Figure 55: Jumper Settings

11.2.2 Connections and Configuration

1. Connect the *RelayNode8* to the *NextMove Breakout* using a suitable CAN cable.
2. Connect the *RelayNode8* to a 24Vdc supply.
3. Make sure that *NextMove PC* terminator is fitted – jumper link M on the PCB. Refer to *CAN Peripherals Installation Manual*.
4. Load *NextMove WorkBench* and start the *terminal emulator*. For further details on using *NextMove WorkBench* refer to *CAN Peripherals Installation Manual*.
5. Load MINT by selecting *Load Application* from the *File* menu. The terminal emulator displays the MINT sign-on message:

   ```
   MINT for NextMove PC.  Version 3.8.
   Copyright …
   ```

6. To add the *RelayNode8* to the network, at the MINT command line type:

   ```
   NODETYPE.7 = _relaynode8
   ```

   where **`_relaynode8`** is a MINT constant which can be used in place of the number **`3`**. This is the signature for *RelayNode 8* and informs MINT that node 7 is a Relay Node.

7. The LED on the *RelayNode8* will start to flash green, approximately once every half-second. Each flash indicates that the *RelayNode8* is participating in CAN activity.

8. Each time a CAN Peripheral is added, a *node live* event occurs. To clear the node live event, type the following at the MINT command line:

   ```
   VIEW CANSTATUS
   ```

   This will monitor the CAN Bus, reporting any events or errors until **[Ctrl]+[E]** is typed. Reading the event will clear it.

9. To confirm that *NextMove PC* is able to communicate with the *RelayNode8* at the MINT command line type:

   ```
   ? NODELIVE.7
   ```

   MINT will reply with the value **1** (true), which indicates that the node with ID 7 is *live*[1]. A reply of **0** would indicate it was *dead*[2].

10. It can also be confirmed that MINT refer tos the correct type of node. At the MINT command line type:

    ```
    VIEW NODELIVE
    ```

    NextMove PC will display the complete list of all 63 nodes, indicating type and whether *live* or *dead*. Node 7 will be shown as being live **L** and a *RelayNode 8*.

11. The outputs are now free to be controlled. For example, to turn output 3 *on* enter the command:

    ```
    REMOTEOUT.7.3=1
    ```

1 A live node can be seen on the network by NextMove and has been registered using **NODETYPE**.

2 A dead node is a node that although may be powered up, cannot be seen by NextMove. A 'D' next to the node number indicates a dead node. You should check the network wiring to the node and make sure the correct CAN bus channel is selected.

11.3 NextMove PC and CAN Peripherals

11.3.1 Selection of CAN Channel

NextMove PC supports CAN Peripherals on its CAN Bus channel 1.

When connecting CAN Peripherals to a *NextMove PC* controller, CAN Bus channel 1 must be selected on the CAN Peripheral by fitting jumpers JP1 and JP2 to position 1, as shown in Figure 55.

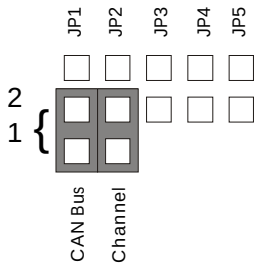


Figure 56: CAN Channel Jumper selection

11.3.2 Selection of CAN Baud Rate

The CAN Baud rate is the rate at which data is transferred over the network. The controller and CAN Peripherals use a default CAN transmission rate of 125 Kbit/s. Although it is possible to alter this, it is unlikely to be necessary. At this baud rate the network may be up to 500m (1640ft) long. Refer to 7.5. for maximum bit rates *vs* cable lengths.

If it is found necessary to change from the default baud rate the CAN Peripherals must be statically configured.

11.3.3 Selection Of Node ID

Each CAN Peripheral must be given a unique Node ID within the network. The Node ID is used to filter out CAN messages that are directed at other nodes and is simply a number. The Node ID is assigned to the node using *NextMove* by a means called static configuration. The rules which govern how Node IDs should be assigned for a *NextMove PC* host are:

- *InputNode 8, OutputNode 8, RelayNode 8, IoNode 24/24* and *KeypadNode* nodes may have any Node ID between 1 and 63.
- No two nodes may have the same Node ID in the same network, as shown in Figure 57.

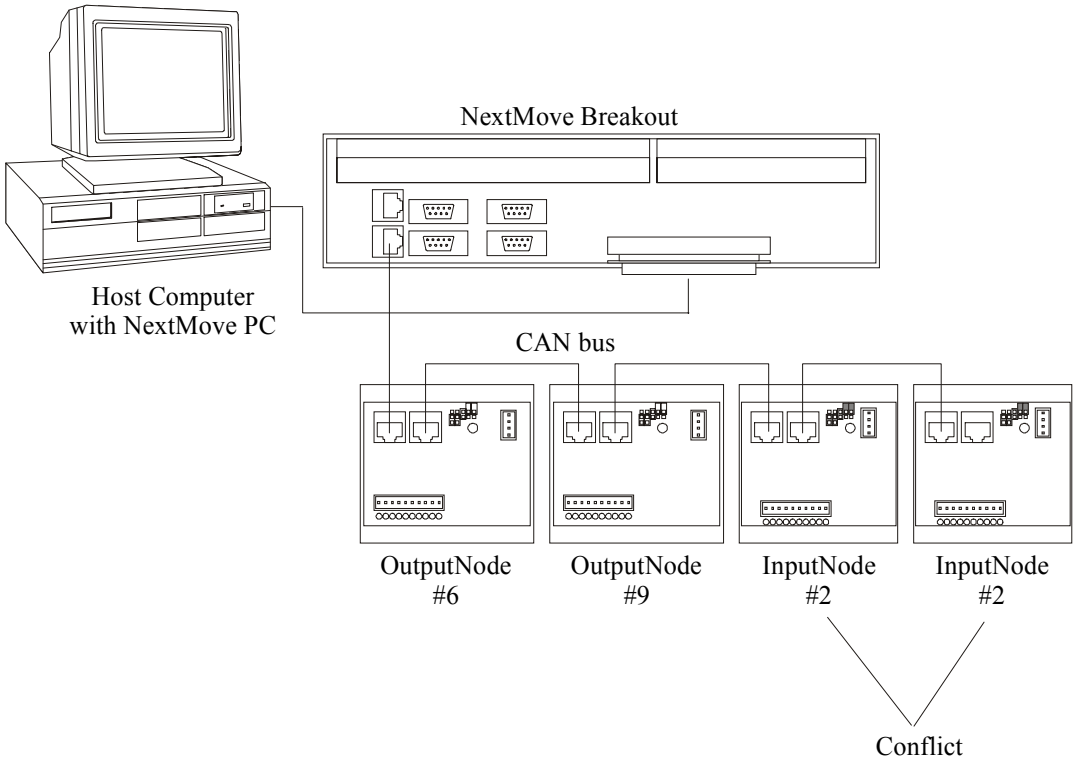


Figure 57: Network with conflicting Node IDs

The nodes do not have to have Node IDs that reflect the order in which they are connected on the network, as shown in Figure 58.

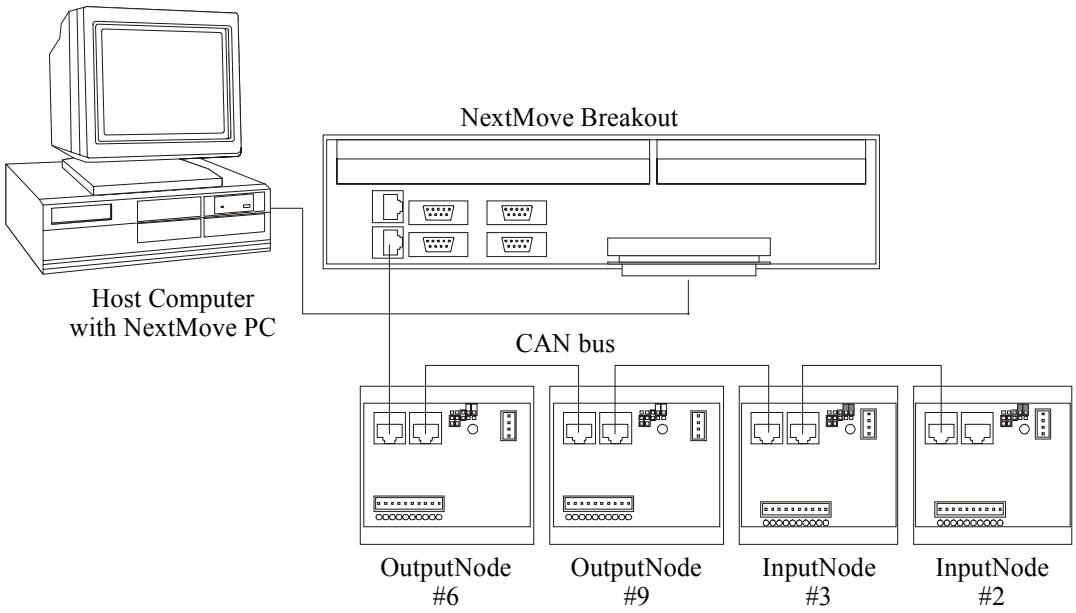


Figure 58: Typical network

To set a Node ID, the node must be statically configured as described in Section 11.3.5.

11.3.4 Network Termination

Termination resistors must be fitted at the ends of the network to reduce signal reflection. The controllers and CAN Peripherals are fitted with termination resistors specifically for this purpose. Refer to Section *CAN Peripherals Installation Manual* for full details.

On *NextMove PC*, the terminator should be selected by fitting Jumper Link M on the PCB. This is fitted as factory default.

On the CAN Peripherals, the terminator is selected by fitting a jumper to JP3, as shown in Figure 59.

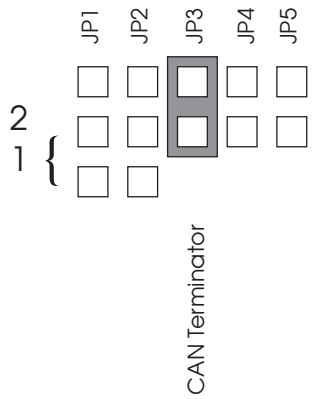


Figure 59: Network Termination Jumper Settings

11.3.5 Static Configuration

An CAN Peripheral has two attributes that may be altered:

- CAN communication baud rate (default 125 kbit/s)
- Node ID. The defaults are shown in the following table:

Node	Default Node ID
InputNode 8	1
OutputNode 8	7
RelayNode 8	7
IoNode 24/24	8
KeypadNode	14

These attributes are altered over the CAN Bus. There must be only two devices in the network - the node to be configured and a CAN based configuration tool (in this case a controller). Because this configuration has to take place with the node removed from the full network it is called *Static Configuration*.

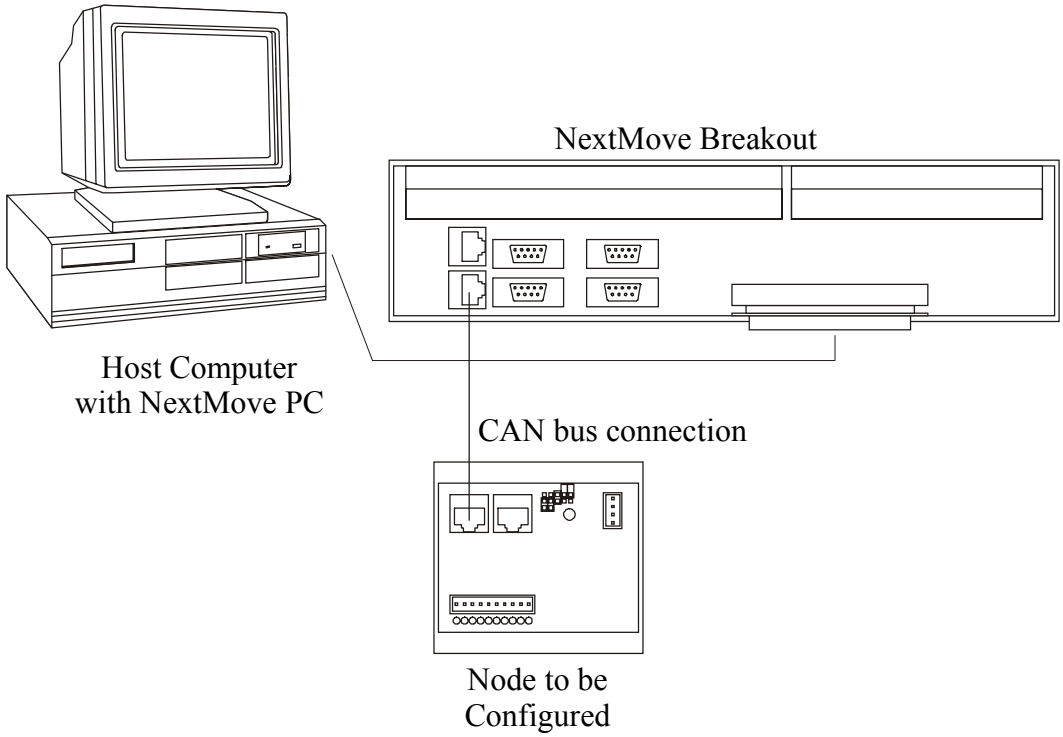


Figure 60: *NextMove PC* **- Node Configuration**

To statically configure a CAN Peripheral:

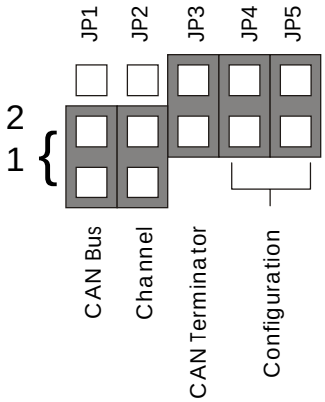


Figure 61: Static Configuration Jumper Position

1. Make sure that jumpers JP1 and JP2 (CAN Bus channel) on the CAN Peripheral are fitted in position 1 (CAN1), as shown in Figure 61.

2. Make sure that jumper JP3 (CAN terminator) on the CAN Peripheral is fitted, as shown in Figure 61.
3. Make sure that jumpers JP4 and JP5 (configuration) on the CAN Peripheral are fitted, as shown in Figure 61.
4. Make a CAN connection between the controller (CAN1 bus) and the CAN Peripheral.
5. Power up the controller.
6. Load the *NextMove WorkBench* and start the terminal emulator. For further details on using *NextMove WorkBench* refer to *CAN Peripherals Installation Manual*.
7. Make sure MINT is running. The terminal emulator displays the MINT sign on message:

   ```
   MINT for NextMove PC.  Version 3.8.
   Copyright …
   ```

8. Power up the CAN Peripheral. The LED on the node will show red.
9. If using a version of MINT lower than 3.8, go to point 10, otherwise at the MINT command line type:

   ```
   REMOTESETUP
   ```

   MINT will respond with the current details of the node. Followed by a prompt for the new Node ID. This is the number that will be used to reference the node in future. Enter a number between 1 and 63 for the Node ID.

   ```
   Node Type = inputnode8
   Current Node Number = 5
   Serial Number = 00000000148420
   Firmware Version = 1.00.b4

   New Node Number ?
   ```

   A prompt for the CAN baud rate will be shown. Enter a valid baud rate.

   ```
   New CAN Baud ?
   ```

   When the node has been configured successfully MINT will display the message:

   ```
   Remote Node is set.
   ```

   Continue from point 14

NOTE: The `REMOTESETUP` keyword automatically changes the baud rate to 125 Kbit/s for configuration and then sets it to the new remote baud rate on completion.

10. Make sure that the controller CAN baud rate is set to 125 Kbit/s by typing at the controller command line:

```
CANBAUD = 125
```

11. At the controller command line type:

```
REMOTENODE = <nodeID>
```

where <**nodeID>** is the number used to reference the node in future.

12. Go to step (13) if using the default baud rate of 125 kbit/s.
13. At the controller command line type:

```
REMOTEBAUD = <baud>
```

where <**baud>** is the CAN baud rate intended for use.

NOTE: The CAN baud will be automatically changed to the new remote baud rate and so must be set back to 125 kbit/s after each call to REMODEBAUD if other CAN Peripherals are to be configured.

14. Power down the CAN Peripheral.
15. Remove jumpers from JP4 and JP5 on the CAN Peripheral.

11.3.6 Normal Operation

When involved in CAN communication the status lights on the CAN Peripherals flash green. The controllers operate a node guarding procedure in which all nodes are regularly sent a CAN message. This procedure is seen on the CAN Peripheral as a flash of the green LED, approximately once every half-second.

For further details on the operation of the CAN status LED, refer to *CAN Peripherals Installation Manual*.

11.4 An Example Network

To illustrate the steps required in putting a CAN network together, consider a network comprising:

- A *NextMove PC* controller
- 2 *InputNode 8* nodes
- 2 *OutputNode 8* nodes
- 1 *RelayNode 8* node

- 1 *IoNode 24/24* node
- 1 *KeypadNode* node

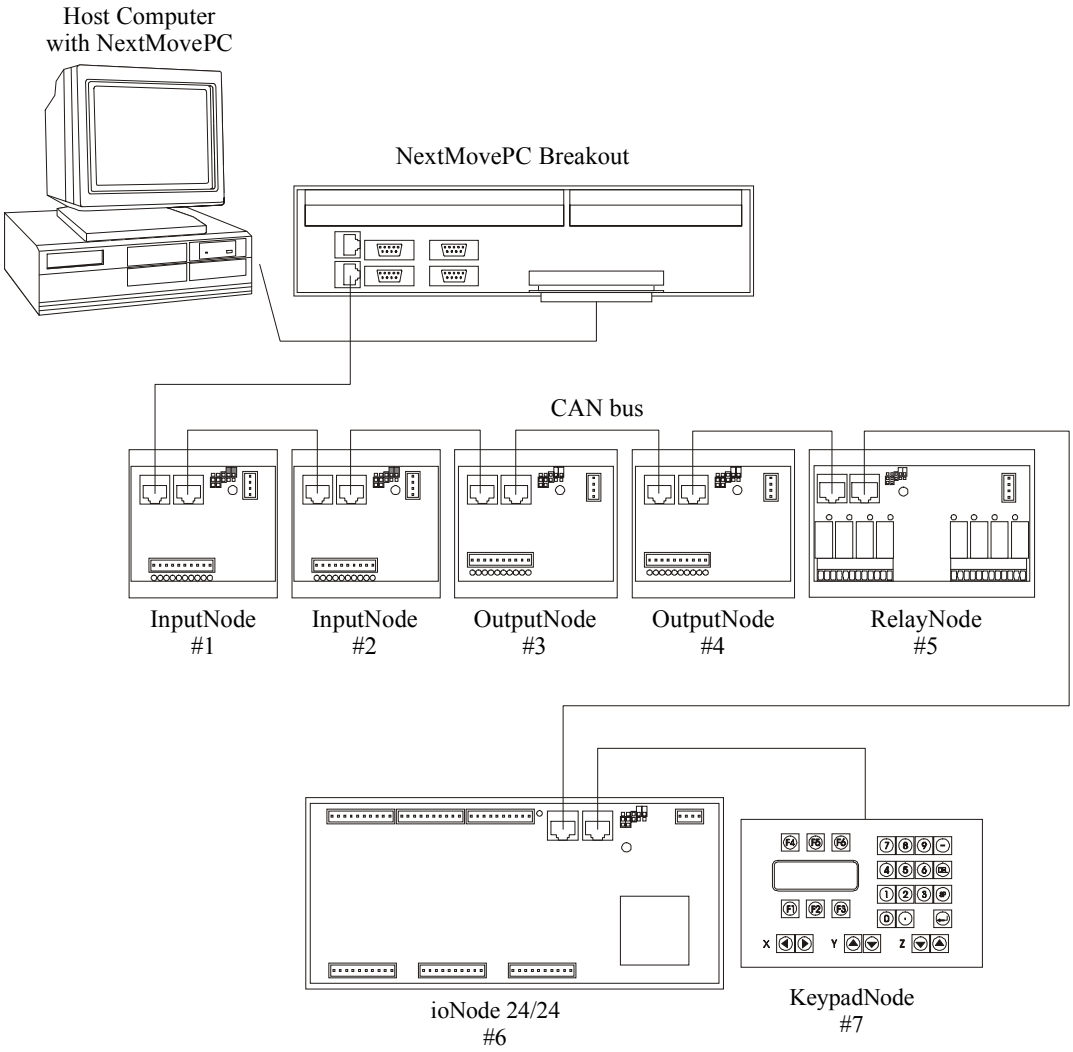


Figure 62: Example multi-node network

The nodes are daisy-chained together with the controller at one end of the network and *the KeypadNode* node at the other end of the network. The controller is housed a short distance from the machine. The CAN Peripherals are distributed over the machine locally to their respective actuators and sensors. Each node is supplied with 24Vdc from a bus supplying the machine. The CAN baud rate used is the default 125 Kbit/s.

1. Statically configure the two *InputNode 8* nodes with Node IDs 1 and 2 (refer to Section 11.3.5).
2. Statically configure the two *OutputNode 8* nodes with Node IDs 3 and 4 (refer to Section 11.3.5).
3. Statically configure the *RelayNode 8* node with Node ID 5 (refer to Section 11.3.5).
4. Statically configure the *IoNode 24/24* node with Node ID 6 (refer to Section 11.3.5).
5. Statically configure the *KeypadNode* node with Node ID 7 (refer to Section 11.3.5).
6. Terminate the network:
   - ◊ On *NextMove PC*, the terminator should be selected by fitting Jumper Link M on the PCB (this is fitted as factory default).
   - ◊ If the *KeypadNode* is at the end of the network its terminator should be selected by fitting a jumper to JP3.
   - ◊ The remaining nodes should have their terminator switched out by removing the jumper on JP3.
7. Position the CAN Peripherals and controller on the machine.
8. Power up the system.

11.5 Using A *KeypadNode*

When connecting a *KeypadNode* to the CAN[3] bus, the node must be added to the network as usual using the **NODETYPE** keyword. In addition to this, the **KEYPADNODE** keyword must be used to inform MINT as to the node identifier of the *KeypadNode* and the terminal channel used to communicate with this node. This is because *NextMove* allows up to 4 *KeypadNode*s to be connected to the bus. Valid terminal channels for *KeypadNode*s are **_tmLCD1**, **_tmLCD2**, **_tmLCD3** and **_tmLCD4**.

Assuming the *KeypadNode* is configured as node 14 (default value), the following code is used to add the node to the network on channel **_tmLCD1**:

MINT Example:

```
NODETYPE.14 = _keypad   : REM Add node to bus
PAUSE NODELIVE.14       : REM Wait for node to be live
KEYPADNODE._tmLCD1 = 14 : REM Set node 14 as the keypad node
```

3 Make sure the correct CAN channel is selected on *Keypad*Node.

Both **_keypad** and **_tmLCD1** are predefined constants in MINT.

The **PAUSE NODELIVE** command is required to force a wait until the node is live before the **KEYPADNODE** keyword is executed. If this is not implemented, an error will occur as there will be a small delay while the node becomes live.

It is now possible to communicate with the device. The **TERMINAL** keyword should be used to turn on the appropriate terminal devices:

```
TERMINAL = _tmLCD1 OR _tmRS232
PRINT "Hello"
```

Hello should appear on both the terminal running WorkBench and the LCD display of *KeypadNode*.

The keyword **TERMINAL** is used to set the terminal input and output channels for *NextMove*. By default, it is assigned to communicate with all supported terminal devices, therefore allowing all terminal keywords to be used with the four *KeypadNode*s. Also, any keypresses on the *KeypadNode*s will be echoed to the host display and vice-versa.

The **VIEW** keyword can be used in MINT to refer to if the *KeypadNode* has been located:

```
VIEW NODELIVE
```

The *KeypadNode* should be displayed next to its node number. An **L** indicates that the node is live[4], whereas a **D** indicates it is dead[5].

Once the *KeypadNode* has been configured onto the bus, the MINT terminal keywords such as **PRINT**, **INKEY**, **LOCATE** and **CLS** can be used.

Refer to *CAN Peripherals Installation Manual* for more details on *KeypadNode*.

4 A live node can be seen on the network by NextMove and has been registered using **NODETYPE**.

5 A dead node is a node that although may be powered up, cannot be seen by NextMove. A **D** next to the node number indicates a dead node. Check the network wiring to the node and make sure the correct CAN bus channel is selected.

11.6 MINT Keyword Summary

Keyword	Description
Configuration Keywords	
CANBAUD/CB	set the controller CAN baud rate
NODETYPE/NT	add/remove a CAN node to the network
REMOTEBAUD/RB	set the CAN baud rate used by a remote node
REMOTEDEBOUNCE/RD	set/read the node input debounce time
REMOTEINPUTACTIVELEVEL/RIA	set active state of digital inputs on a remote CAN node
REMOTEOUTPUTACTIVELEVEL/ROA	set active state of digital outputs on remote CAN node output active level
REMOTENODE/RN	set the node identifier (Node ID) of a remote CAN node
REMOTESETUP	set the node identifier (Node ID) and the CAN baud rate of a remote CAN node.
Error Keywords	
CANSTATUS/CST	read the cause of an asynchronous CAN event/error
NODELIVE/NL	refer to if a CAN node on the bus is live or dead
REMOTEESTOP/RES	control emergency stop state of a remote CAN node
REMOTEOUTPUTERROR/ROE	read/reset the digital outputs on a remote CAN node that are in error
REMOTERESET/RR	force a remote CAN node to do a software reset
REMOTESTATUS/RS	read/clear the status register on a remote CAN node
STATUSNODE/SN	read Node ID of a node that caused an asynchronous CAN event or error
Input/Output Keywords	
REMOTEIN/RI	reads the state of digital inputs from a remote CAN node
REMOTEOUT/RO	read/set the state of digital outputs on a remote CAN node

MINT Programmer's Toolkit

12

The MINT Programmer's Toolkit is a set of PC applications and example files (Visual Basic, Borland Delphi etc) for use with NextMove PC. The main component of the MINT Programmer's Toolkit is the NextMove WorkBench. This chapter concentrates on the functionality of the NextMove WorkBench.

- The NextMove WorkBench.
- Software oscilloscope.
- DOS utilities for NextMove PC.

The MINT Programmer's Toolkit is a set of PC applications and examples for use with *NextMove PC*. The main component of the MINT Programmer's Toolkit is *WorkBench*.

The software is supplied on CD-ROM. To install the software insert the CD-ROM into the PC. This action initiates the automatic loading of a 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`**.

12.1 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 Windows 3.x:

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

Windows NT does not support the *NextMove WorkBench*

Once started the default screen will show:

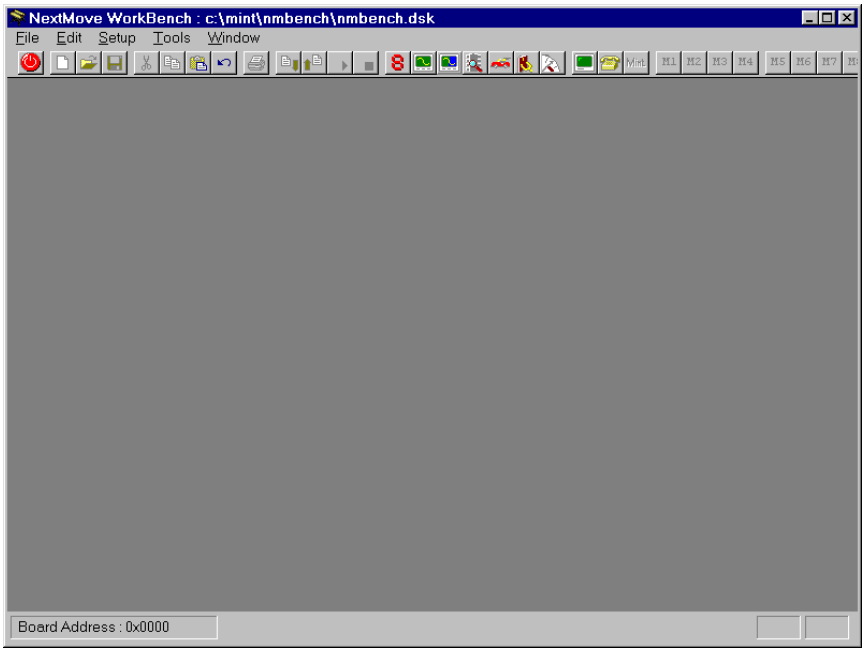


Figure 63: Default *WorkBench* Window

12.2 The Main Toolbar

Many of the tools in *WorkBench* can be launched from the main toolbar. The tools are described in more detail later in the manual.

Reset Nextmove: Resets the *NextMovePC* Controller.

Download File: if an editor is selected, the editor file will be downloaded. Otherwise a dialog box will prompt for the file to download.

Upload File: if an editor is selected, the file will be uploaded to that editor. If an editor is not selected, a dialog box will appear, prompting an upload to an open editor. A new editor for uploading should be created, or 'save to file' initiated.

Send MINT **`RUN`** command.

Send *Comms abort* and MINT **break** (**[Ctrl]+[E]**) command.

Axis WatchWindow: auto-update of *position*, *velocity*, *following error*, *mode* and *error*

Software Oscilloscope: graphing axis variables

Software Oscilloscope Settings

DPR WatchWindow: watching locations of DPR

Motion WatchWindow: auto-update of *position*, *velocity*, *following error* on axis

Digital I/O: Real-time status of digital I/O

Analog Inputs: Real-time status of analog inputs

Terminal: for communication with the MINT command line.

MINT Comms: for communication with a running MINT program using the protected mode communications array.

to Execute macros 1 through to 10.

12.3 Selecting the Board Address

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. (The *NextMove PC* address is selectable using jumpers - refer to Section 5.1 of this manual for details). 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 should be entered by selecting from the list of valid (hexadecimal) addresses by clicking on the *down* arrow adjacent to *NextMove Address (Hex)*. Refer to Figure 64.

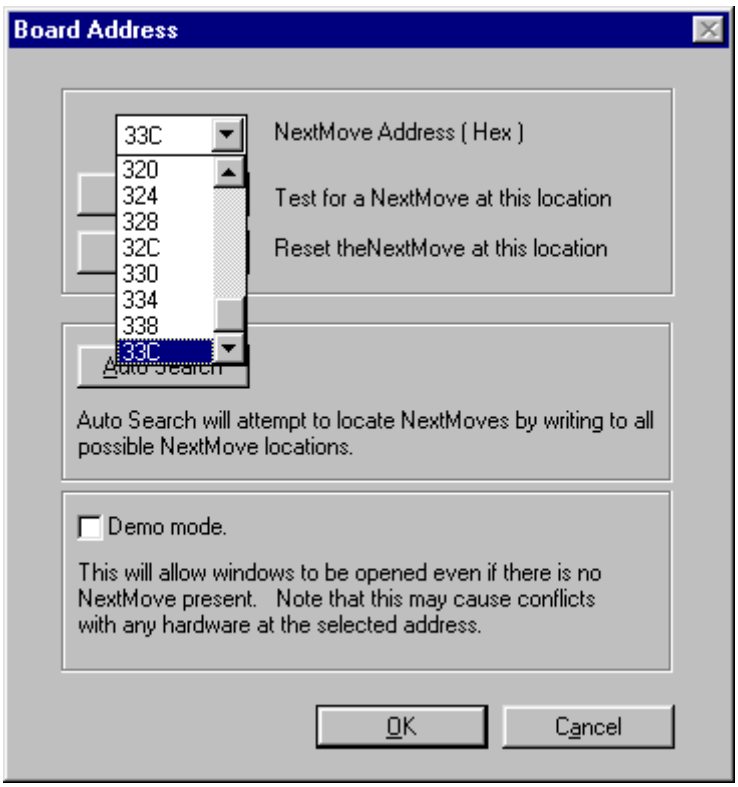


Figure 64: NextMove Address Window

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.

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.

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.

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*.

12.4 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*.

To download an application to *NextMove PC*:

Select *Download Application* from the *File* menu. This will show the *Download Application* dialogue box which enables an application file to be selected. Clicking *OK* will download the application to the controller.

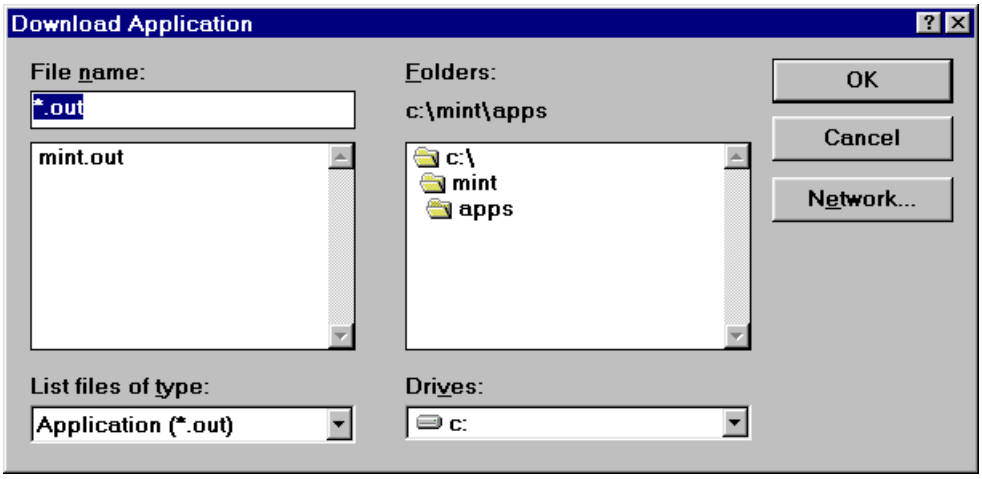


Figure 65: Download Application

12.5 Terminal

The Terminal window provides access to the MINT command line.

To enable the Terminal Window:

1. From the *Tools* menu select *Termina*l, or
2. Click the icon on the toolbar

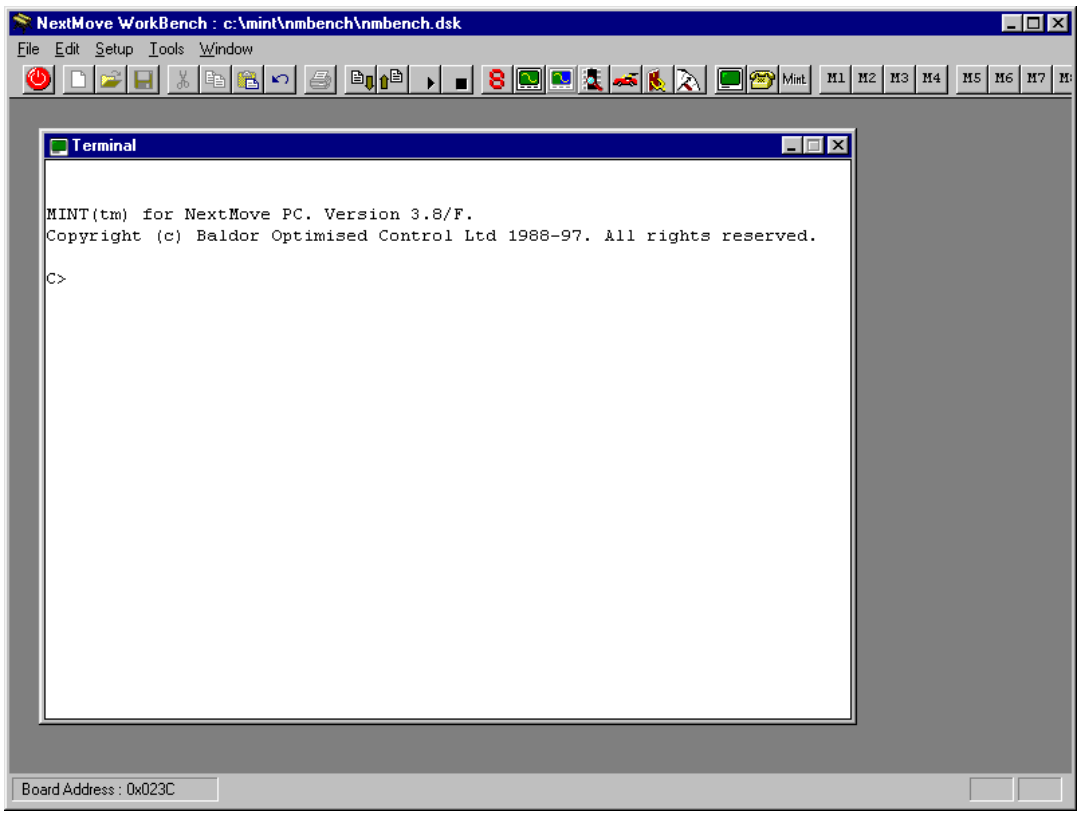


Figure 66: Terminal window

The terminal can be resized up to a maximum size of 50 lines by 80 columns either by dragging the border, or:

1. From the *Setup* menu select *Terminal*

The *Select Terminal Size* Dialog Box enables the terminal to be sized by entering values for **Lines (1-50)** or **Columns (10-80).** Refer to Figure 67.

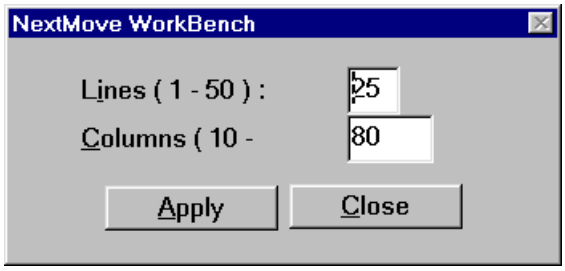


Figure 67: Select Terminal Size Dialog Box

12.6 Program/Configuration Editors

A *WorkBench* editor with *cut and paste* and *search and replace* facilities is available from the *File* menu. Either a new file can be selected or an existing file opened. Separate editors are supported for both the MINT *program* and MINT *Configuration buffer*s.

There are several ways to initiate an editor:

1. From the *File* menu, select *New File*, and then choose from *Program*, *Config* or *Text*.
2. To open an existing file, from the *File* menu, select *Open File*, and then choose from *Program*, *Config* or *Text*. Directories may then be searched for the required file.
3. To upload a file into an editor from the controller, from the *File* menu, select *Upload File*, or from the toolbar click .

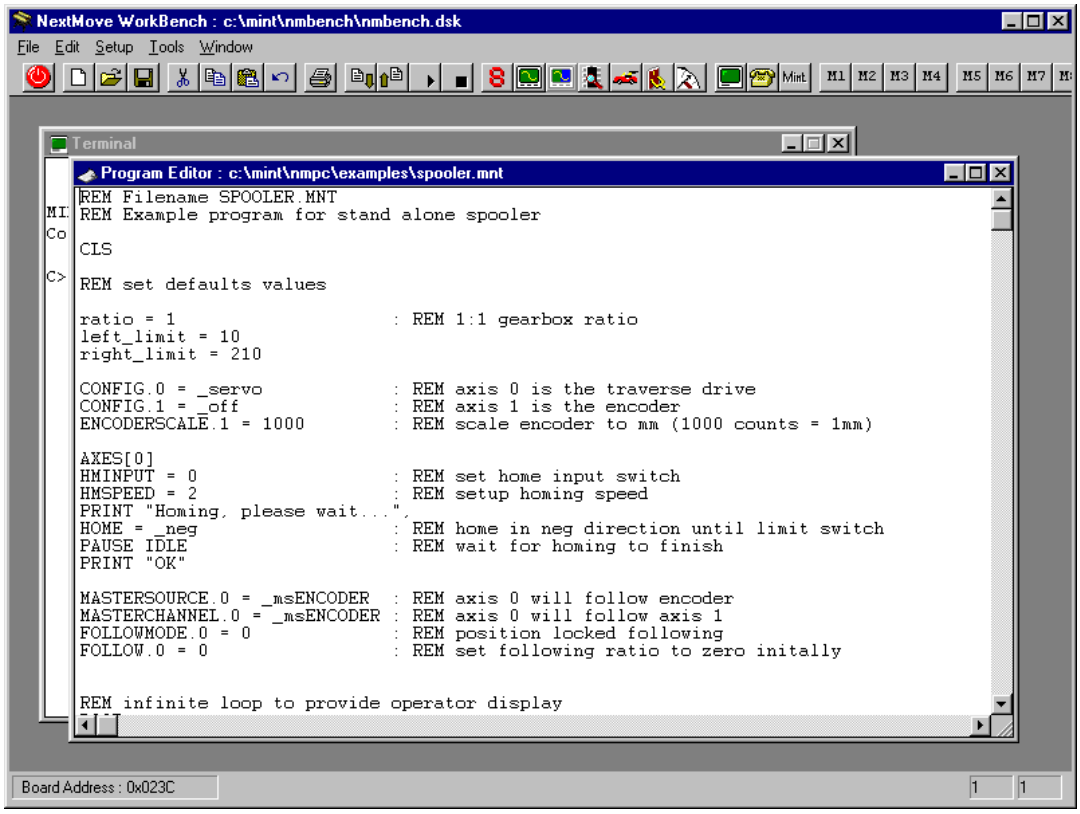


Figure 68: Editor Window

The standard Windows *Cut*, *Copy* and *Paste* functions are available both on the toolbar and in the *Edit* menu.

Other editor functions include:

- Downloading the files to the controller by clicking the icon on the button bar. By using the *Setup* ⇒ *Squash* menu, files can be compressed on download. Alternatively, selecting *Edit* ⇒ *Squash* will squash the current editor file to a new file. Full details of *Squash* can be found in section 12.7.
- Uploading the files from the controller into the current menu. *WorkBench* will automatically upload the correct file from the controller depending upon which editor window is open.
- Ability for MINT files to be dragged from Windows *Program Manager* or *Explorer* onto the *WorkBench* desktop. This will automatically open the editor.
- Printing from the editor using the *Print* icon on the tool bar. The file is printed to the currently installed Windows printer.

12.7 Squash

Squash allows a program to be compacted by removing redundant characters – blank lines for example. It is possible to save from an editor in the squashed format, or to have *WorkBench* squash each time it downloads a file.

Squash can perform several operations on a file. However, some of them can make the code unreadable. Therefore, the options are individually selectable.

To initiate the *Setup Squash Parameters* dialog box:

1. From the *Setup* menu, select *Squash.*

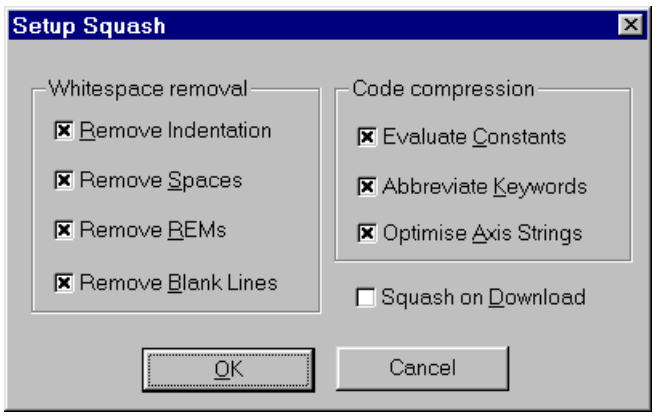


Figure 69: Setup *Squash* Window - Parameters

Whitespace removal section

- **Remove Indentation –** Clicking this check box removes spaces from the start of lines.
- **Remove Spaces –** Clicking this check box removes any unnecessary spaces other than at the start of a line.
- **Remove REMs –** Clicking this check box removes all **`REM`** statements.
- **Remove Blank Lines –** Clicking this check box removes any blank lines.

Code compression section

- **Evaluate Constants –** Clicking this check box replaces any MINT constants with their numeric values. For example, it replaces **`_servo`** with **`1`**.
- **Abbreviate Keywords –** Clicking this check box replaces any longhand MINT keywords with their two or three letter forms.
- **Optimise Axis Strings –** Clicking this check box will replace any single axis statements in the square bracket form with the dot form. For example: **`MA[0]`** will be squashed to **`MA.0`**

General section

- **Squash on Download** – Clicking this check box Squashes each file as it is downloaded. This allows the file to be edited in the editor in its full 'readable' form, but will download a compact version of the code.

12.8 Macros

Ten macro buttons are located on the right hand side of the *WorkBench* toolbar. Each macro allows a text string to be sent to the controller.

To initiate the setup of *Macro Entry*:

1. From the *Setup* menu, select *Macros*

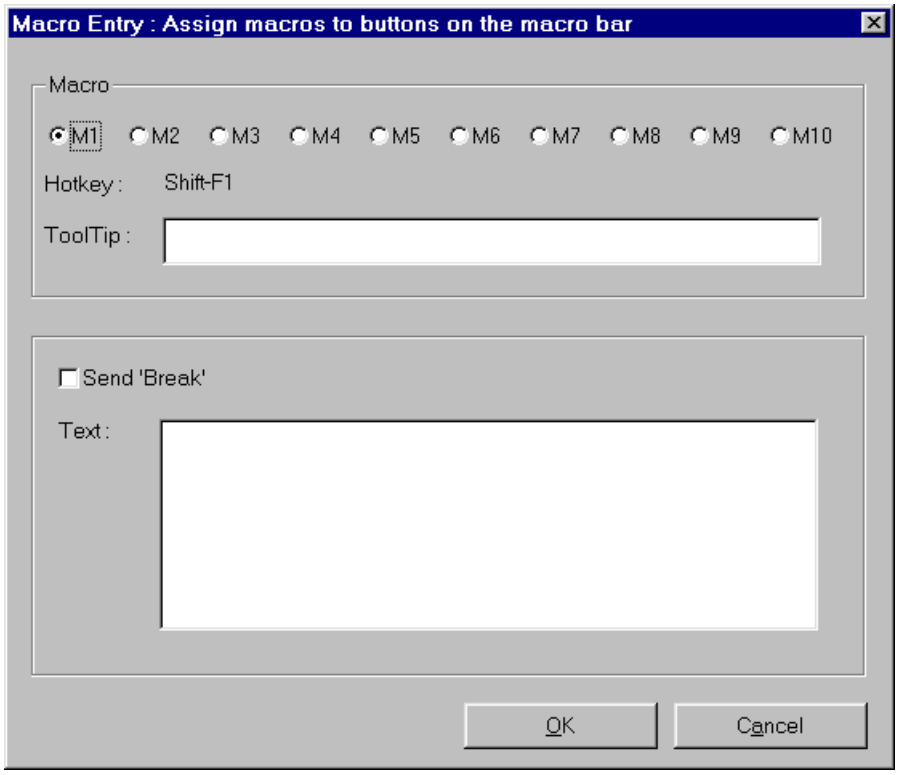


Figure 70: Macro Entry Window

The *Macro Entry* dialog box allows selection of:

- **M1** to **M10 –** Click a macro number radio button to select the macro to edit. (The **Hotkey** field also shows how to achieve this with keyboard shortcut. This hotkey cannot be changed.)
- **ToolTip –** This field displays the text that will be shown when the mouse is placed over a button.
- **Send 'Abort Comms' –** This check box is used to abort protected mode communications (**COMMSON**) on the controller before sending the macro text.

- **Send 'Break' –** This check box is used to send the **break** command (**`[Ctrl]+[E]`**) to the controller before sending the macro text.
- **Send at 9600 baud –** This check box is used to specify that the string be sent at 9600 baud. This is useful when using the macro to change that baud rate when the controller has been reset. Refer to the example above.
- **Text –** This field allows up to 255 characters of free text to be sent. The *Enter* key should be pressed at the end of the text if *Enter* is to be sent to the controller.

12.9 Setting up Gains

The *NextMove WorkBench* allows to use to modify system gains at any point using the *System Gains* window. This is useful during the initial set-up and tuning of motors as it allows the gains to be changed while the controller is running a program, hence the controller can be performing positional moves and the gains can be modified 'on the fly'.

- From the *Set-up* menu choose *Gains*.

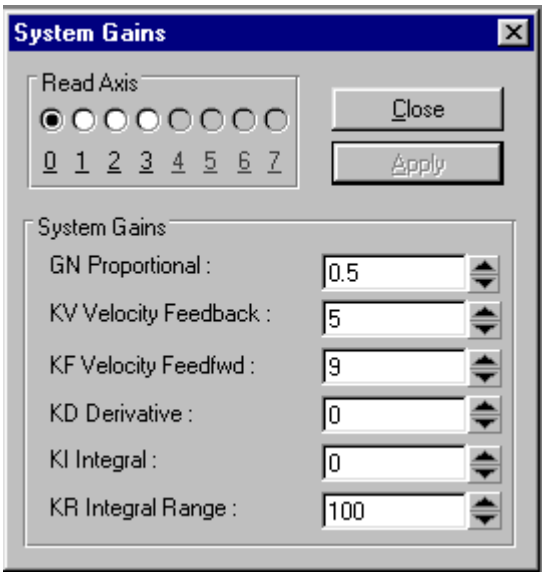


Figure 71: System Gains

To change the gain terms for an axis, select the appropriate axis using the radio buttons and modify the parameters as required. The changes to these parameters to not take effect until the *Apply* button has been clicked.

12.10 Motion WatchWindow

- From the *Tool* menu choose *Motion Watch Window,* or
- Click on the toolbar.

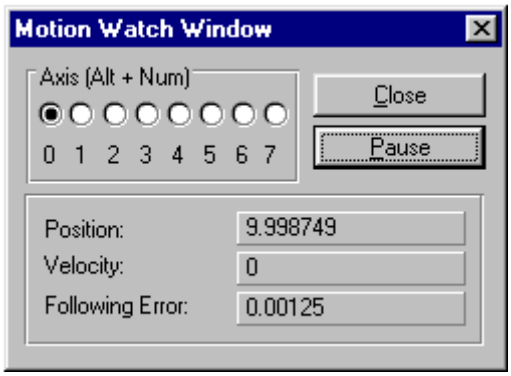


Figure 72: Motion WatchWindow

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 250 line encoder, the position should change by ±1000 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.

12.11 The Comms Window

The MINT Comms Protocol is supported through the use of the *Comms Window*, allowing the debugging of a MINT program. The Comms Protocol is discussed in *MINT for NextMove Programmers Manual*.

To initiate the *Comms Window*:

- From the *Tools* menu, select *COMMS Window,* or
- Click on the toolbar.

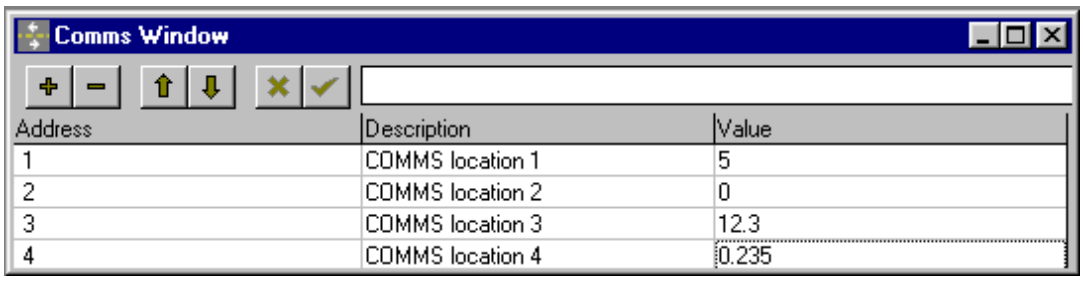


Figure 73: Comms Window

Locations are added by clicking the button, and removed by clicking the button. The **Address** field shows the *comms array index* and the **Description** field is free form text. The data within the **Value** field is then updated periodically

It is possible to change the value of a comms location by clicking on the **Value** field and editing it in the **Edit** field. The button should be clicked to update the value and for it to be written back to the controller. Note that *Enter* will not write back the value. The button can be clicked to cancel the value.

Lines in the grid can be rearranged by selecting the required comms location to be moved. The buttons can then be clicked to move it up or down.

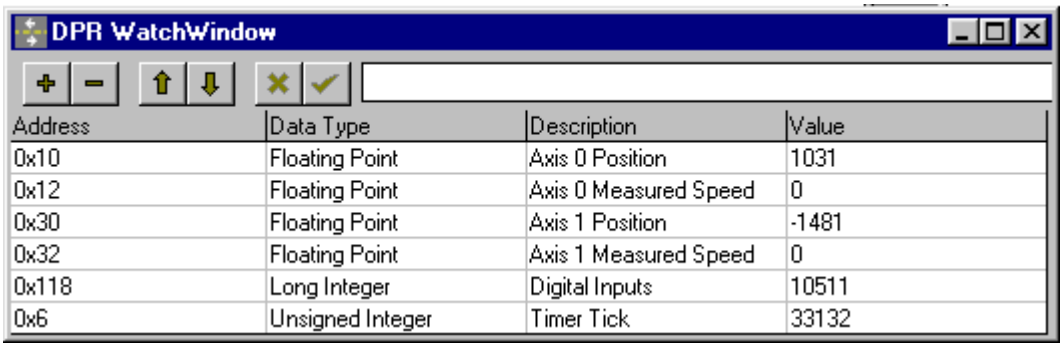
Notes:

1. The *Comms Window* polls the controller on a regular basis. This will slow the MINT program running on the controller as it will have to spend a high percentage of its time processing the comms requests. The more comms locations added to the window, the slower the MINT execution.
2. To pause the *Comms Window*, the *Terminal Window* should be selected. Only one *Terminal Window* and *Comms Window* can be active at any one time. Therefore, selecting one disables the other.

12.12 Dual Port Ram WatchWindow

To initiate the *DPR WatchWindow*:

- From the *Tools* menu, select *DPR WatchWindow,* or
- Click on the toolbar.



Address	Data Type	Description	Value
0x10	Floating Point	Axis 0 Position	1031
0x12	Floating Point	Axis 0 Measured Speed	0
0x30	Floating Point	Axis 1 Position	-1481
0x32	Floating Point	Axis 1 Measured Speed	0
0x118	Long Integer	Digital Inputs	10511
0x6	Unsigned Integer	Timer Tick	33132

Figure 74: DPR WachWindow

Locations are added by clicking the button, and removed by clicking the button. The **Address, Data Type** and **Description** field shows static information about the location of DPR being monitored. The **Value** shows the data at the DPR address, this is updated periodically.

It is possible to change the value of a DPR location by clicking on the **Value** field and editing it in the **Edit** field. The button should be clicked to update the value and for it to be written back to the controller. Note that *Enter* will not write back the value. The button can be clicked to cancel the value.

Lines in the grid can be rearranged by selecting the required comms location to be moved. The buttons can then be clicked to move it up or down.

All of the items in the *DPR WatchWindow* can be graphed using the *Software Oscilloscope*.

12.13 Software Oscilloscope

The software oscilloscope provides a method of graphing axis variables, this is very useful during the tuning of motors. In order for a value to be graphed it must first be added to the *DPR WatchWindow*. Any location within DPR currently in the DPR watch window can be plotted by the software oscilloscope.

To initiate the *Software Oscilloscope*:

- From the *Tools* menu, select *Software Oscilloscope,* or
- Click on the toolbar.

This opens up the *Sample Parameters* dialogue box which enable the selection of plots and graph style.

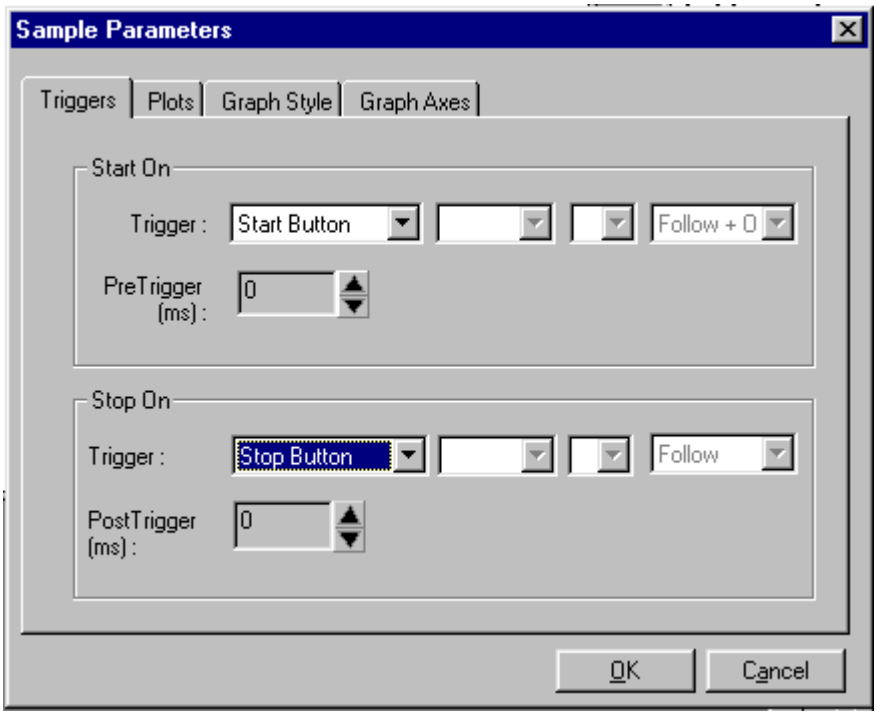


Figure 75: Software Oscilloscope parameters

- ***Triggers***: Selection of a start and stop trigger and pre and post trigger times. By default the graph is started and stopped manually by clicking the start and stop buttons. However the graph can be made to trigger on a motion type.
- ***Plots***: Select items to be plotted, this gives the option of plotting any selection of items which are currently in the *DPR WatchWindow*.
- ***Graph Style*** : Specification of line widths and colours for all of the plots
- ***Graph Axes*** : Selection of auto scaling or scaling to one specific plot, also the selection of the X-axis variable (by default this is time) to enable X/Y plots to be generated.

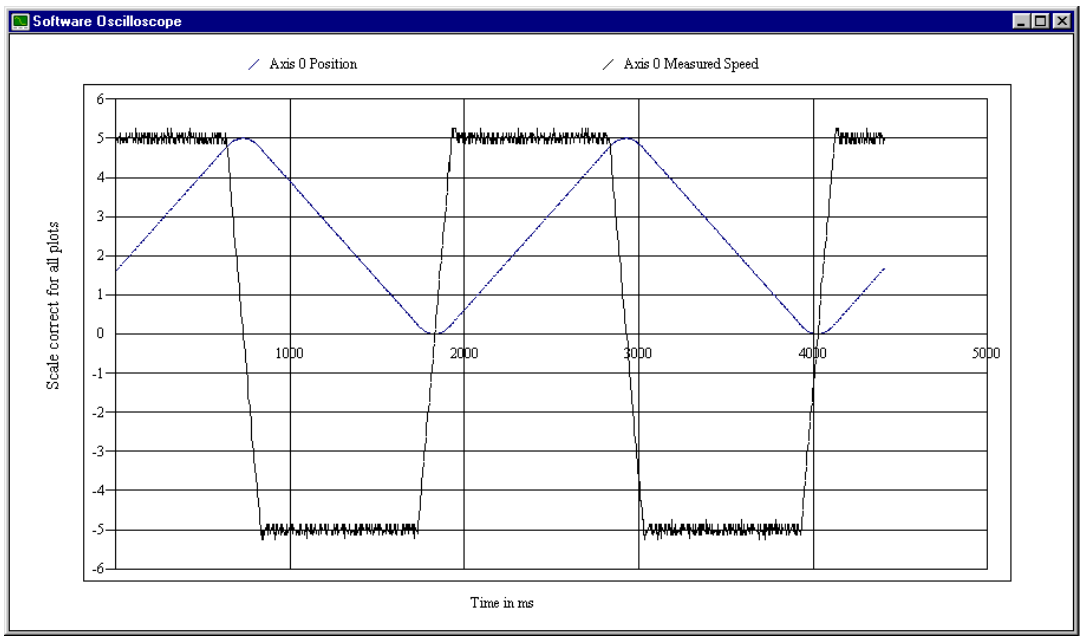


Figure 76: Software Oscilloscope

Once the graph has been plotted the *Graph Style* can be changed by clicking on the icon from the toolbar. This opens the *Sample Parameters* dialogue box but only the *Graph Style* tab is enabled.

12.14 Axis WatchWindow

The *Axis WatchWindow* monitors the position, velocity, following error, mode and error on all eight axes. It allows one of the above parameters to displayed for all eight axes, this can be changed by clicking on the appropriate tab. The values are updated periodically to give a real time monitor.

To initiate the *Axis WatchWindow*:

- From the *Tools* menu, select *Axis WatchWindow,* or
- Click on the toolbar.

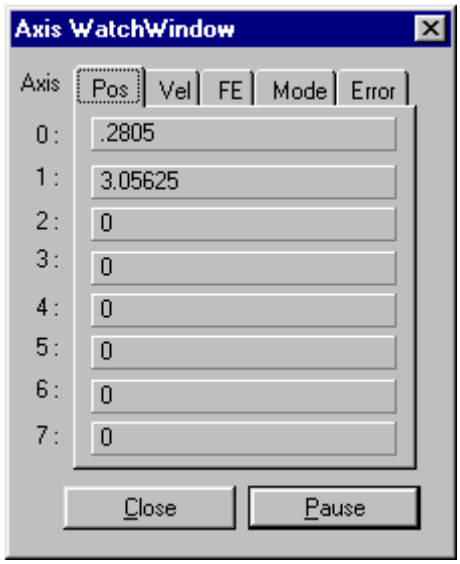


Figure 77: Axis WatchWindow

Double clicking on an axis value will display the *Motion WatchWindow* for that axis.

12.15 Digital I/O

The *Digital I/O* windows monitors the state of all digital inputs and outputs including the relay. The values are updated periodically to give a real time monitor. The state of the outputs and relay may be changed by clicking the relevant checkbox.

To initiate the *Digital I/O*:

- From the *Tools* menu, select *Digital I/O,* or
- Click on the toolbar.

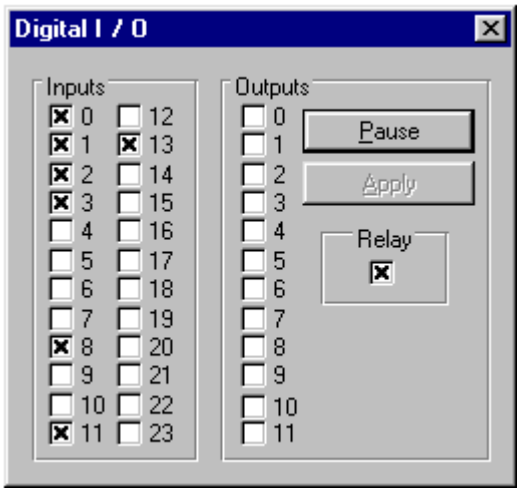


Figure 78: Digital I/O

12.16 Analog Inputs

The *Analog Inputs* windows monitors the state of selected analog input channels. The values are updated periodically to give a real time monitor.

To initiate the *Analog Inputs*:

- From the *Tools* menu, select *Analog Inputs,* or
- Click on the toolbar.

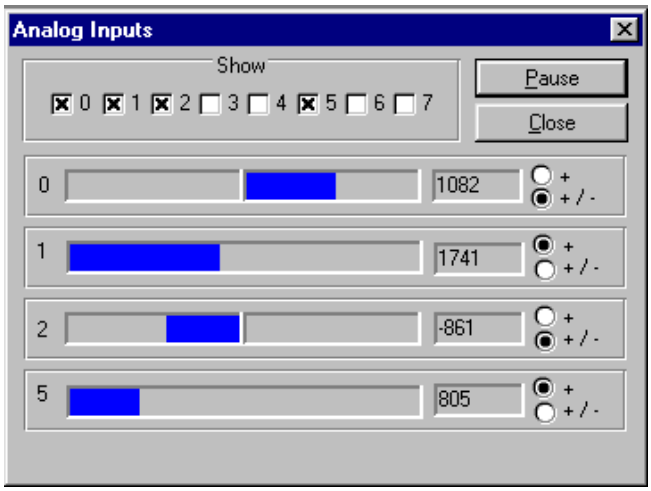


Figure 79: Analog Inputs

12.17 Project Files

WorkBench uses Project Files to record the following information:

- Open editors.
- Current board address
- Locations displayed in the *Comms Window*.
- Terminal settings.

This is particularly useful if many elements have been added to the *Comms Window*. In order to save a project, the *File* ⇒ *Save Project* menu option should be selected. The information is saved to a file with a .**CTW** extension. Once saved, the .**CTW** file can be added to the Taskbar in Windows 95. Selecting the icon will start *Workbench* with the desktop settings. There are several ways to open a desktop:

- If *WorkBench* is started normally, the most recent desktop is used.
- Create an icon for the desktop (as above).

Drag the desktop file from the *File Manager* or *Windows Explorer* and drop it on to *WorkBench*.

12.18 DOS Utilities

There are various DOS utilities supplied with the MINT Programmer Toolkit these are installed into the directory \MINT\DOUTILS\NMPC. Details of these utilities are given below:

NMAUTO allows the downloading of application, program and configuration files to a *NextMove PC* for DOS. Ideal for adding to an AUTOEXEC.BAT file for automatic configuration of a *NextMove PC* on boot up of a PC.

NMFIND will locate any *NextMove PC*'s and display their I/O address.

NMFIND will write to all I/O locations supported by NextMove PC. If there are other device which use these locations there functionality may be affected and this may cause the PC to lock up.

NMINFO will display details on the boot loader and memory details of a *NextMove PC* at a specified I/O address. The NextMove PC must be in reset for NMINFO to be able to function correctly.

NMMEM is used to test the memory on the *NextMove PC* card. It will also test memory on the memory expansion card if fitted.

NMRESET will reset a NextMove PC at a specific I/O address,

NMRUN allows an application file to be downloaded to a *NextMove PC*.

NMVER will locate and display the version number from a valid *NextMove PC* or *NextMove BX* firmware file. This utility works with .OUT, .HEX and .LIB files.

FILE_UD enables file uploading and downloading of program and configuration files to and from a controller.

Syntax information about any of the above utilities can be obtained by typing the command name at the DOS command line with no parameters.

Trouble Shooting Guide

13

This chapter covers common problems found when using the NextMove PC controller.

Symptom	Check
Cannot communicate with the controller.	• Verify that the *WorkBench* program is loaded and set-up correctly. The board address must be supplied and an application (e.g. **MINT.OUT**) downloaded. • Check the card is firmly seated in its socket in the computer and this socket is of the correct type. • Check that card address jumper is set correctly. • Check that the green LED on the card is flashing (approx. 0.5Hz).
Controller appears to be powered-up and working but will not cause motors to turn over.	• Check that the connections between motor and controller are correct. (Use **TQ[0,1,2,3] = 100**; to set all axis numbers to +10V and verify that this voltage appears on the demand input to the amplifier - remember to ensure that the motors will not cause any physical damage when doing this). • Ensure that the amplifier is enabled and working when the controller is not in error. (When the controller is first powered up the amplifiers should be disabled if there is no program running (there is often an LED on the front of the amplifier to indicate status). Typing **RESET** at the **C>** prompt should cause the amplifiers to be enabled.) • Check that the *configuration file* is loaded into controller and has been **RUN**, or that the servo loop gains are correctly set-up. (Check the *Gains* window or type **PRINT GN** to verify that the controller proportional gain **GN** is not zero and refer to servo-system set-up).

Symptom	Check
Motor runs off uncontrollably when controller is switched on.	• Check that the encoders are connected to controller and are functioning correctly. (Use a dual trace oscilloscope to display both channels of the encoder simultaneously). • Check that the amplifiers are connected correctly to the controller and that with zero demand there is 0V at the amplifier demand input. (Type **`TQ[0,1,2,3] = 0;`** to set all demand outputs to 0V and verify that this voltage appears at the output from the controller. A voltage of +10V or -10V indicates that the controller analog output is damaged.) • Verify that the amplifiers are correctly set-up and that the motor does not move with 0V on the demand input. • Verify that the controller and amplifier are correctly grounded to a common earth point.
Motor runs off uncontrollably when controller is switched on and servo loop gains are applied or a when a move is set in progress. Motor then stops after a short time.	• Check that encoder 0 and demand 0 are connected to the same axes of motion; repeat for axis 1 and 2. • Check amplifier connection is correct with respect to polarity of amplifier demand. (Try swapping the **demand+/command+** and **demand-/demand-** connections to the amplifier; note: this is not possible with some amplifiers due to ground reference problems in which case you need to swap the encoder channels **A** and **B**.) • Check that the maximum following error is set to a reasonable value. For setting up purposes the maximum following error should be set to a high value. Type **`MF[0,1,2,3] = 16000`**; to set all axes to maximum following error of **`16000`** encoder counts, assuming a scale factor of 1)
Motor is under control, but vibrates or overshoots during a move.	• Servo loop gains may be set too high. (Go through the servo system set-up procedure to establish correct gains.)

Symptom	Check
Motor is under control, but when moved to a position and then back to the start it does not return to the same position.	• Check that the encoders channels **A** and **B** are clear signals and that they are correctly wired to the controller. (Use a dual trace oscilloscope to display both channels of the encoder at the controller back plane. Verify that when the motor turns, the two square wave signals are 90 degrees out of phase.) • Check that the encoder signal is free from electrical noise. (Use the oscilloscope as above). Verify that the complimentary outputs on the encoder (if fitted) are correctly wired. • If single ended encoders are fitted to the motors and the signals are noisy, try re-routing the encoder cables to avoid any source of electrical noise (notably the motor power leads). If this fails, the only solution may be to fit encoders with differential line driver outputs. • Ensure that the encoder leads use screened cable and that the screen is attached to the screen connection on the encoder plug **at the controller end only**.) • Verify that the controller and amplifier are correctly grounded to a common earth point.

Bibliography

14

Bibliography

1. *MINT for NextMove Programmers Manual.* Baldor Optimised Control 1998, document reference number: MN1261.
2. *CAN Peripherals Installation Manual*. Baldor Optimised Control 1998, document reference number: MN1255.
3. *MINT Interface Library - Standard and Developer Release.* Baldor Optimised Control 1998, document reference number: MN1259.
4. *Mathematical Method of Calculating System Gains*. Baldor Optimised Control 1998.