7. Introduction to the MINT™ Programming Language

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*.

7.1 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.

7.2 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 6.

7.2.1 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.

7.3 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.

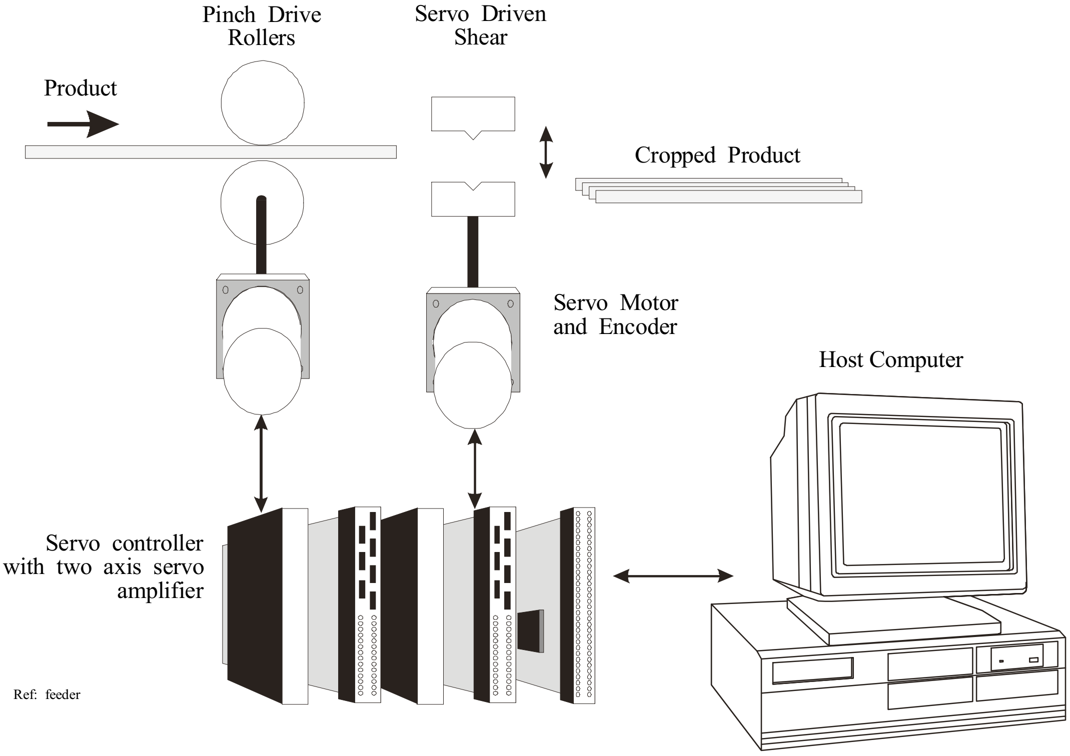


Figure 24: Schematic Diagram of the Cut to Length Feeder

7.3.1 *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 6.4.10, Setting System Gains of this guide for further information.

7.3.2 *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.

7.3.3 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
```

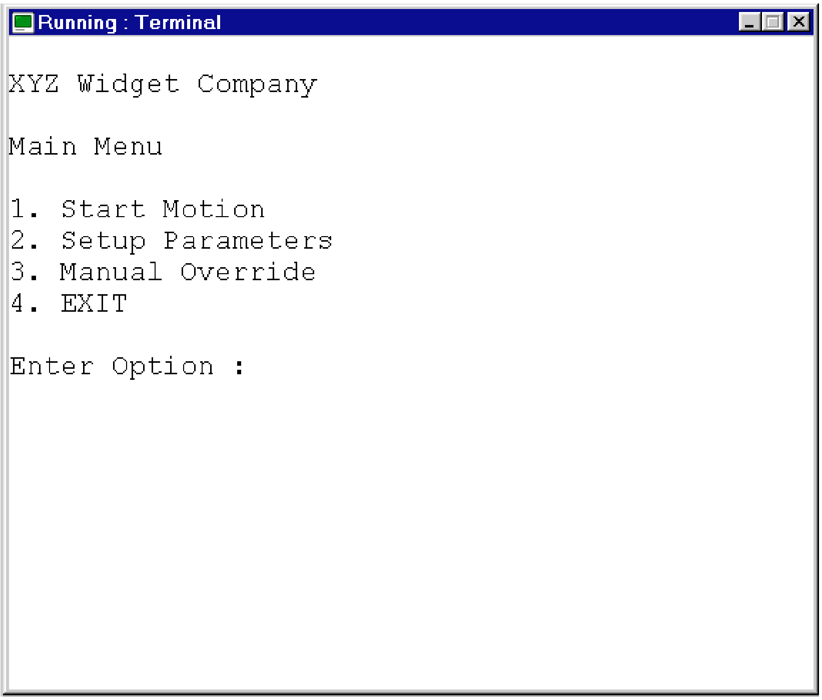
The program is well commented and by making suitable adjustments to the **SCALE** factor and gains in the *Configuration buffer* it should be possible for the program to operate on any system equipped with at least one axis of motion.

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:



```
XYZ Widget Company

Main Menu

1. Start Motion
2. Setup Parameters
3. Manual Override
4. EXIT

Enter Option :
```

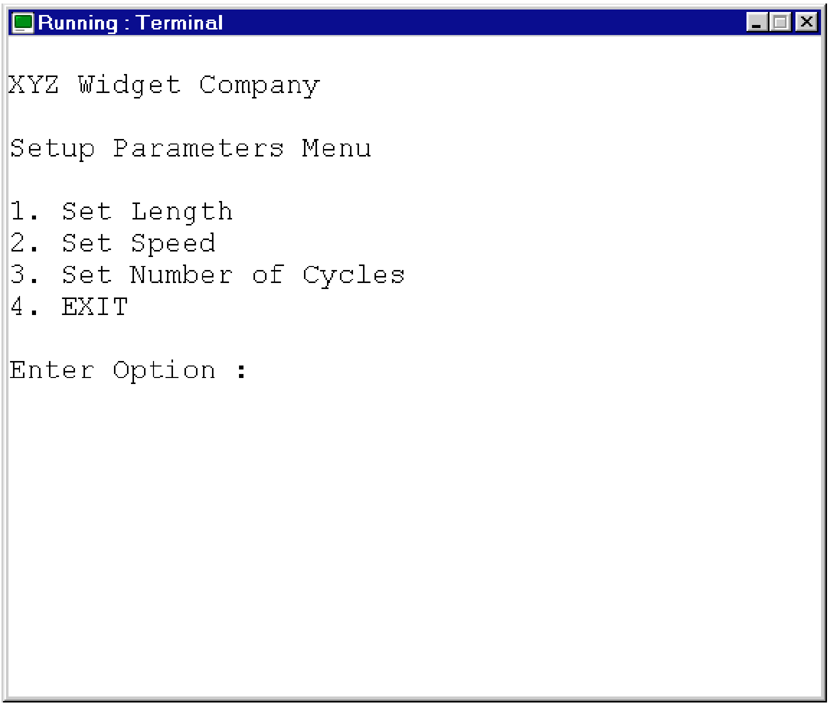
Figure 25: Main Menu

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:



```
XYZ Widget Company

Setup Parameters Menu

1. Set Length
2. Set Speed
3. Set Number of Cycles
4. EXIT

Enter Option :
```

Figure 26: 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.

7.3.4 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.

7.4 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.

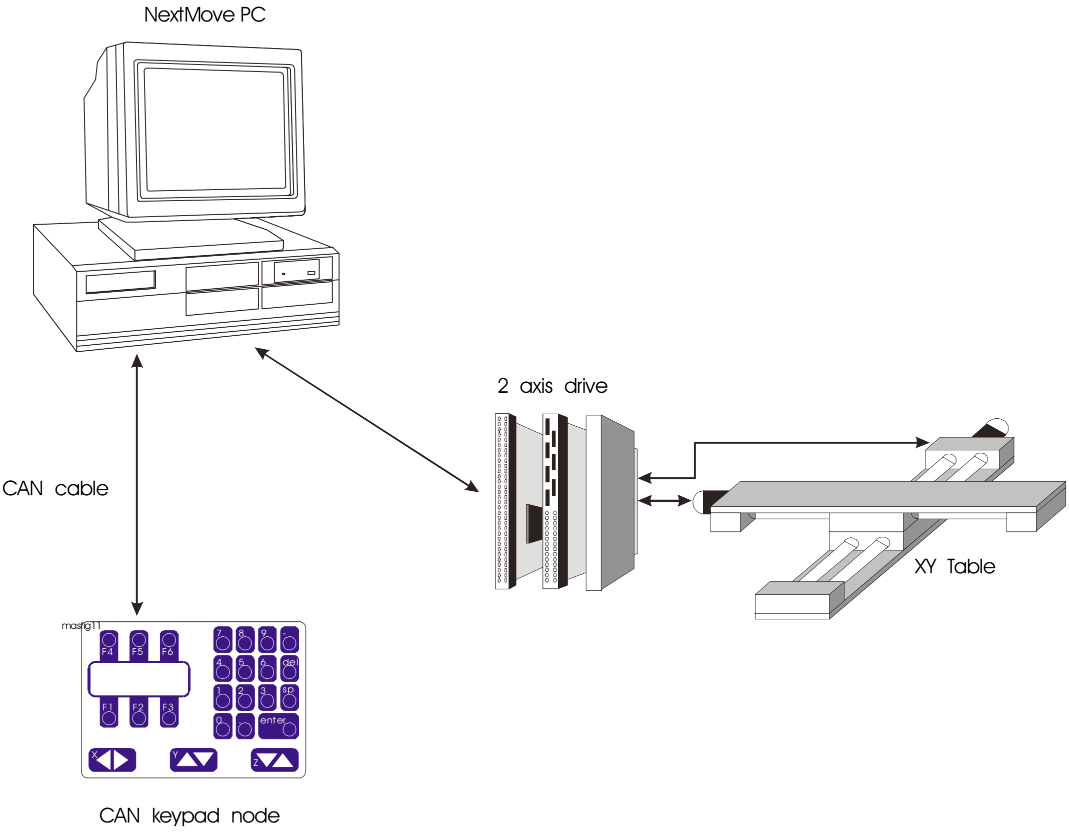


Figure 27: XY Table Control System

🖫 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*.

7.5 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.

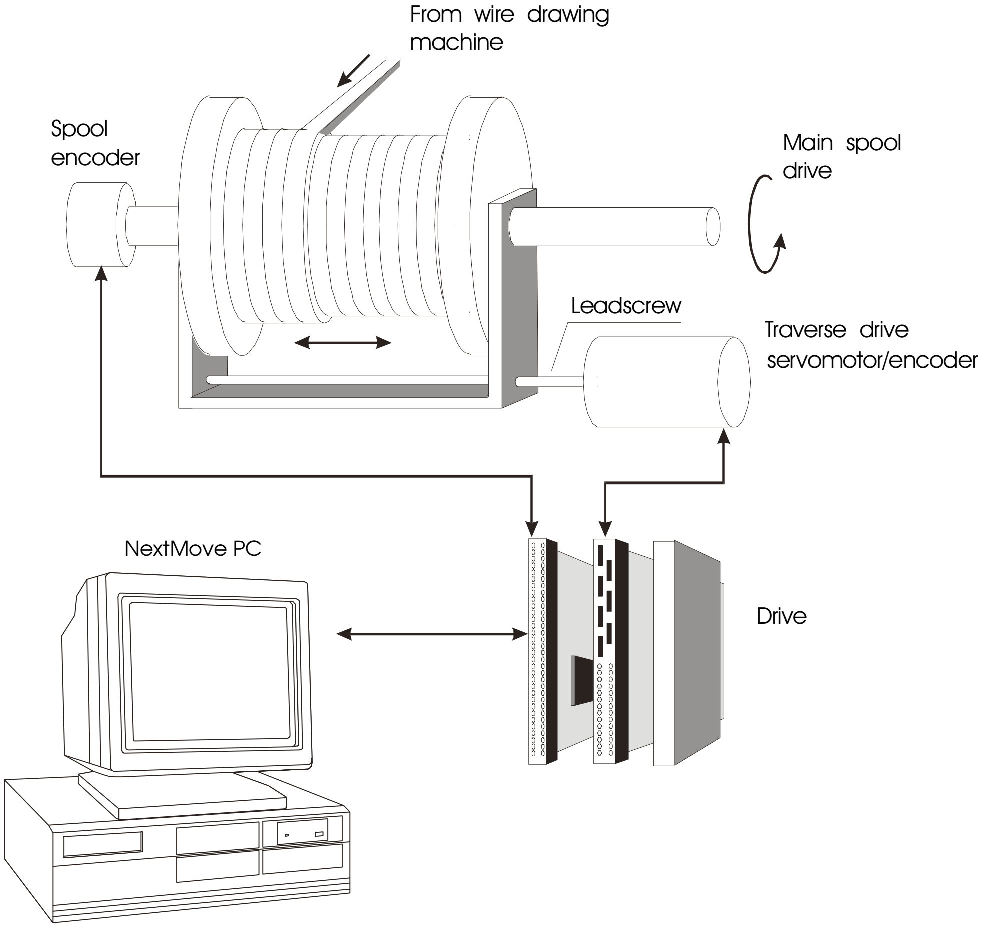


Figure 28: General Layout of Coil Winding Machine

🖫 SPOOLER.MNT

File is in the MINT\NMPC\EXAMPLES directory.

7.5.1 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.

7.5.2 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.

7.6 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.

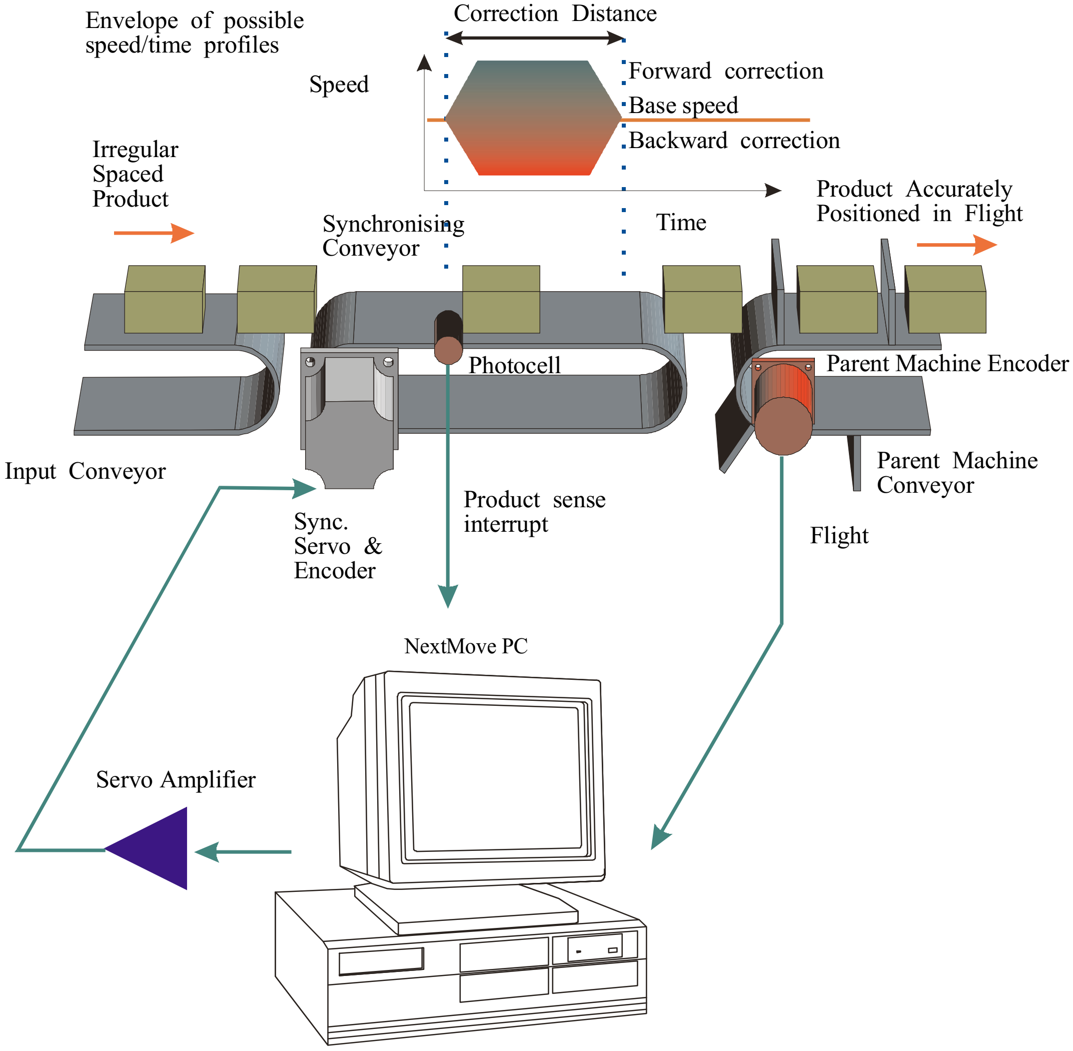


Figure 29: Diagram of Product Infeed Machine

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.

8. Hardware Guide

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

8.1 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

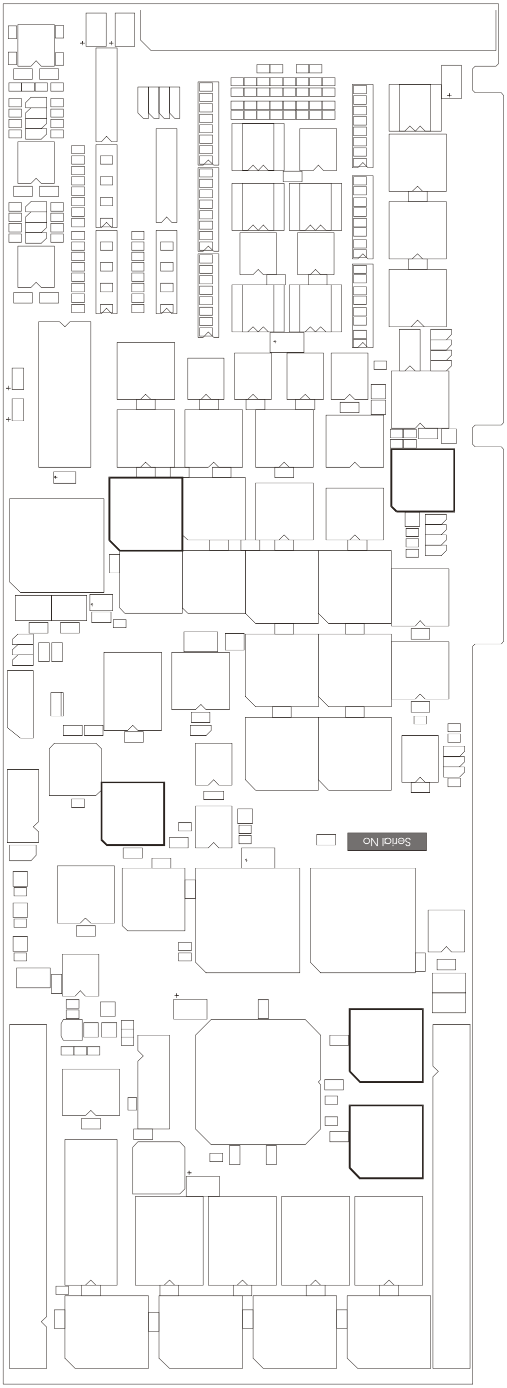


Figure 30: PCB

8.2 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 13.

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

8.3 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 8.3.2) 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 8.5.

8.3.1 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:

	NPN		PNP	
Inputs	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:

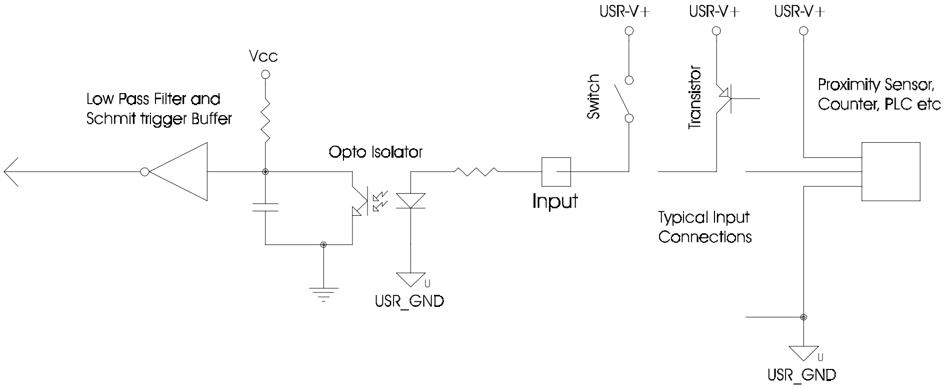


Figure 31: Digital Inputs, PNP configuration

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+**).

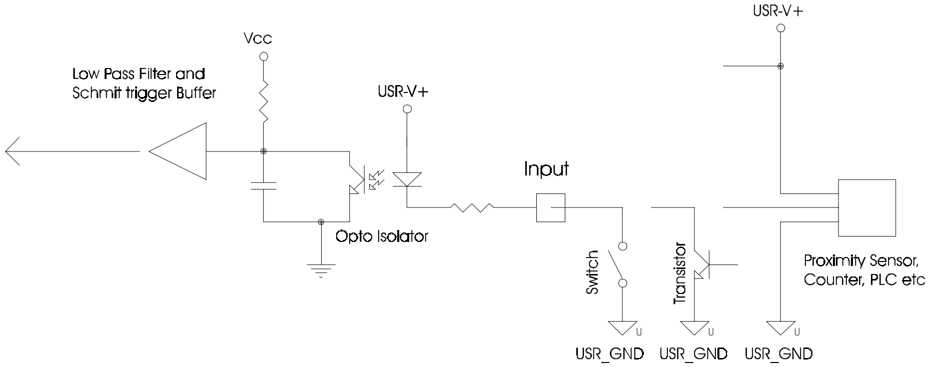


Figure 32:Digital Inputs, NPN configuration

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

8.3.2 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.

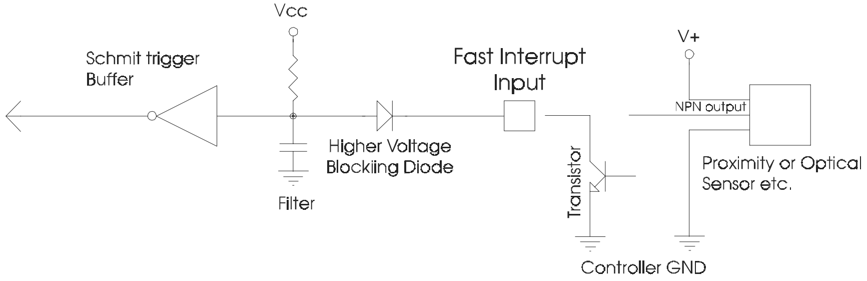


Figure 33: Fast Interrupt Circuit

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

Associated MINT keywords are:

#FASTPOS, FASTPOS, FASTENCODER, FASTAUXENCODER,

8.3.3 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.

8.3.3.1 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:

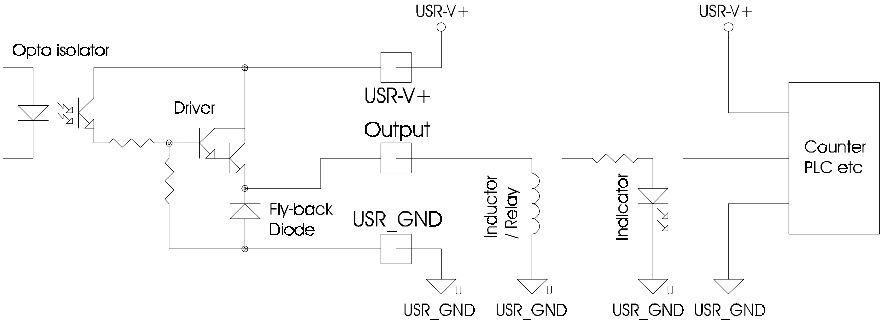


Figure 34: Digital Outputs PNP configuration

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**).

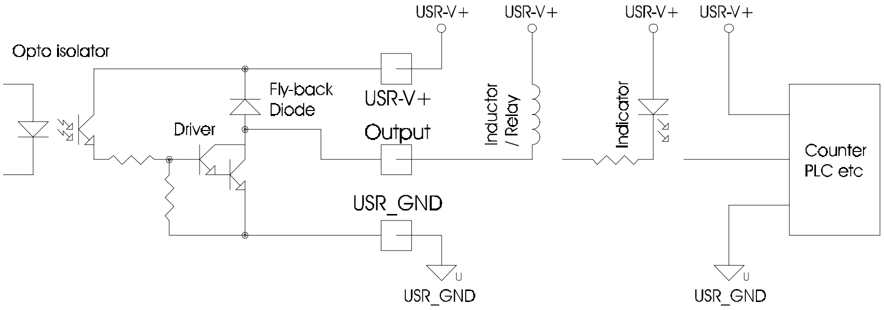


Figure 35: Digital Outputs NPN configuration

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:

	NPN (common is USR-V+)		PNP (common is USR_GND)	
Outputs	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**.

8.3.3.2 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

8.4 Analog I/O

8.4.1 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.

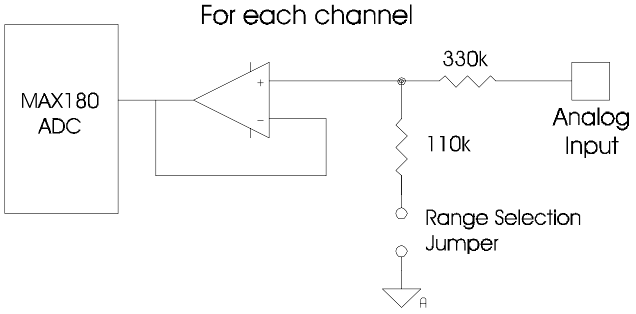


Figure 36: Analog Buffer and Range Selection

8.4.1.1 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:

Channe l	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.

8.4.1.2 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.

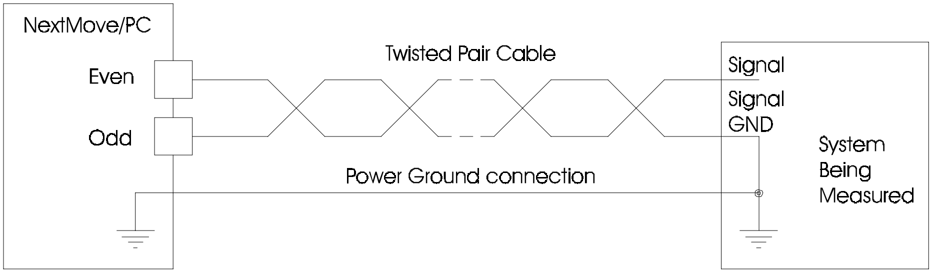


Figure 37: Analog Input, Differential Connection

Associated MINT keywords are:

ADCMODE, ANALOGUEx

8.4.2 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.

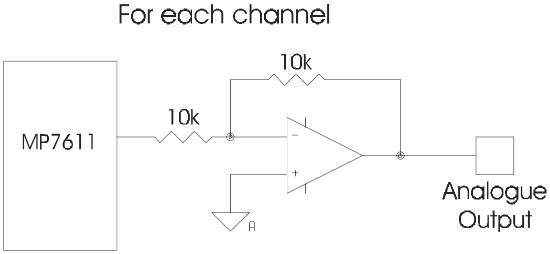


Figure 38: Analog Output buffer

CURRLIMIT, DAC, DACMODE, DEMAND

8.5 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.

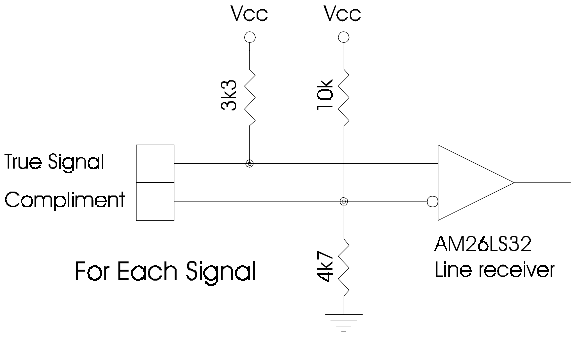


Figure 39: Encoder line receiver interface

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 13 for details).

Associated MINT keywords are:

ENCODER, ENCODERSCALE, ENCODERVEL, ENCODERWRAP

8.6 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

8.7 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.

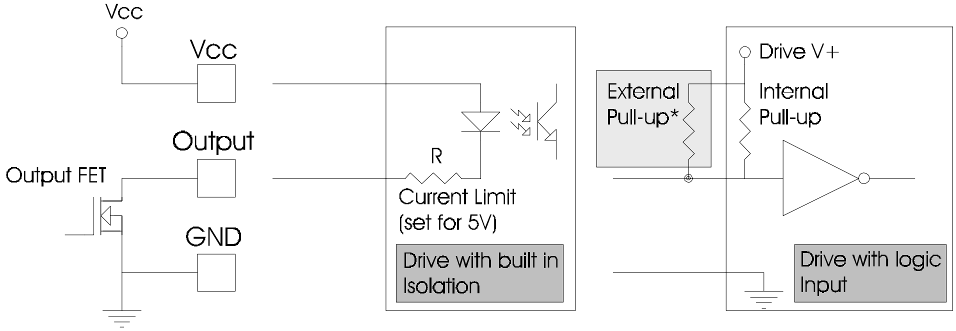
The outputs are all N-channel *mosfets* rated at 100mA@24V and are referred to system ground. The 12 outputs are arranged as four groups of 3 signals whose names and functions are shown in the following table:

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 40: Stepper Drive Connections

Associated MINT keywords are:

BOOST, PULSE, STEPDIRECTION

8.8 Communications

8.8.1 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.

8.8.2 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:

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

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	0.34-0.60mm$_2$
300m-600m (1180-2362ft)	100 kbit/s	< 40mΩ	0.50-0.60mm$_2$
600m-1000m (2362ft-3937)	50 kbit/s	< 26mΩ	0.75-0.80mm$_2$

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 on connector 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

8.9 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.

8.9.1 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.

8.9.2 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.

8.9.3 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.

8.9.4 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.

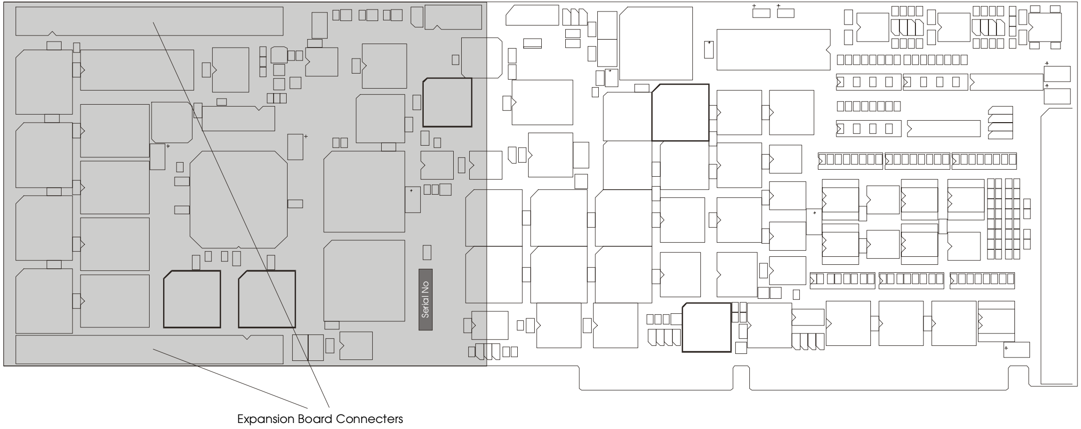
8.10 LEDs

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.

8.11 Expansion Board

Two box headers on the *NextMove PC* controller (as marked in Figure 41) 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*.



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.
4 axes Stepper Expansion Board	OPT011-501	Provides a further 4 axes of stepper control giving a full 8 axes of stepper control

The area occupied by the boards is shown as the shaded area in

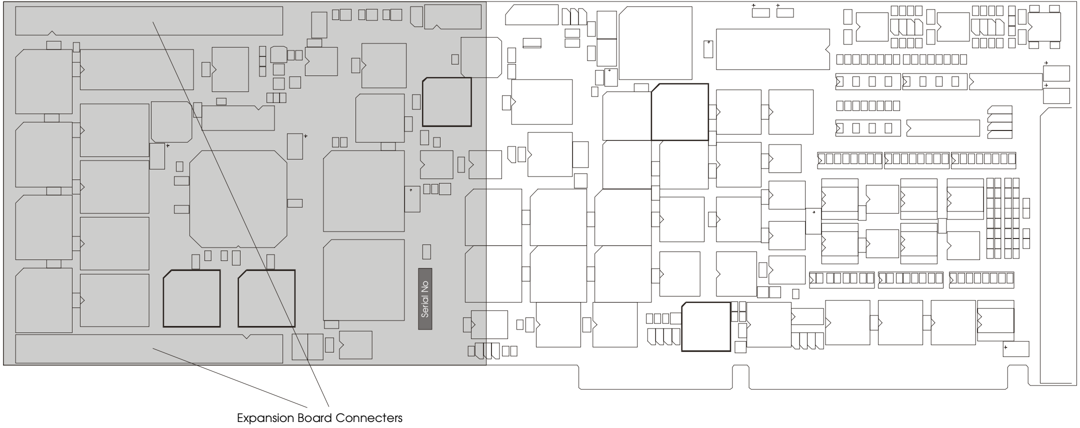


Figure 41. Connections to the motor drives are brought out via a separate slot on the PC.

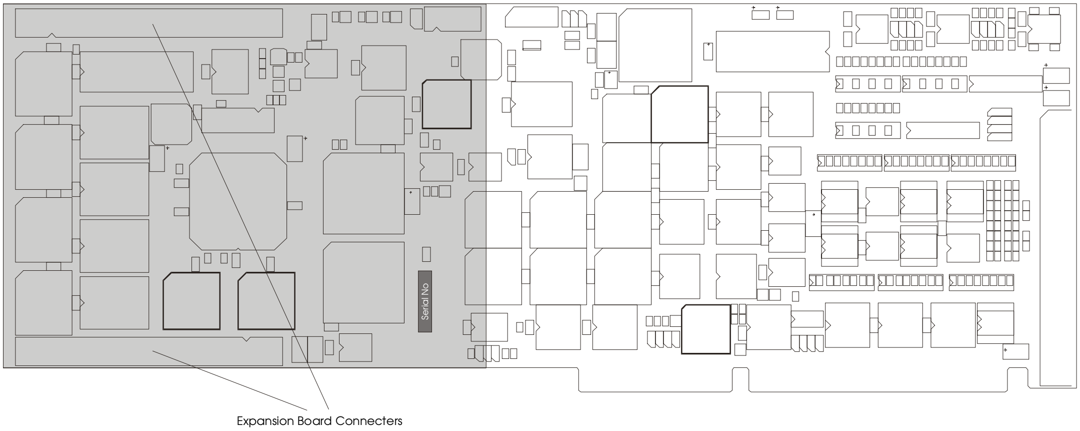


Figure 41: NextMove PC Expansion Board Connections

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*.

8.12 Miscellaneous

8.12.1 Emulator Connection

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

8.12.2 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.

8.12.3 Interrupt Level

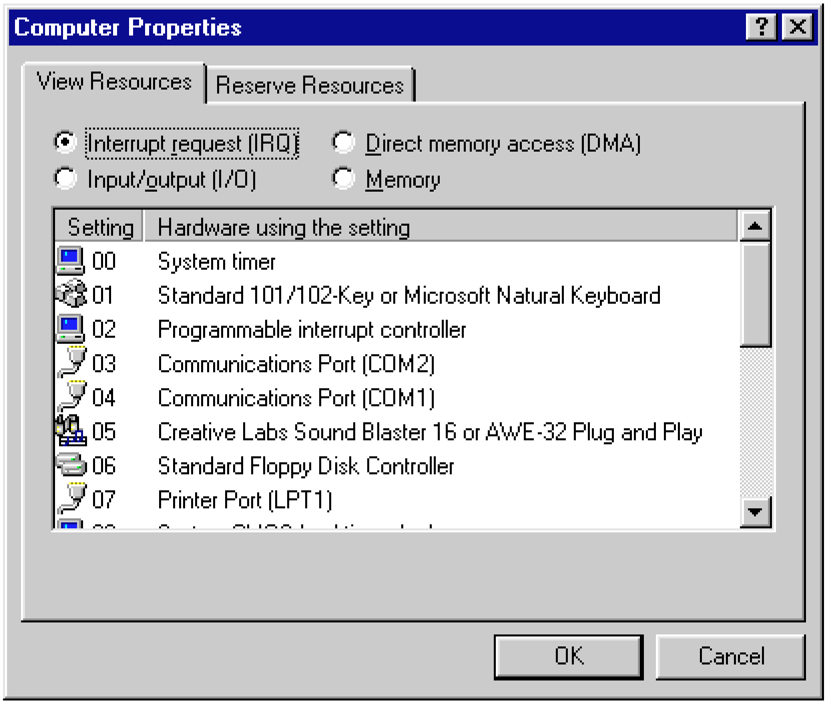
If the PC is running a DOS based host application that requires interrupt handling, then in addition to the I/O address it is necessary to choose a spare interrupt level for use by the *NextMove PC* card.

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

Interrupt level	Typical use	Selected by fitting jumper
IRQ3	Serial Port 2	I
IRQ5	Parallel Port 2	J
IRQ9	Spare	K
IRQ10	Spare	G
IRQ11	Spare	F
IRQ12	Spare	E
IRQ15	Spare	D

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.



● Interrupt request (IRQ)
○ Direct memory access (DMA)
○ Input/output (I/O)
○ Memory

Setting	Hardware using the setting
00	System timer
01	Standard 101/102-Key or Microsoft Natural Keyboard
02	Programmable interrupt controller
03	Communications Port (COM2)
04	Communications Port (COM1)
05	Creative Labs Sound Blaster 16 or AWE-32 Plug and Play
06	Standard Floppy Disk Controller
07	Printer Port (LPT1)

Figure 42: 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.

9. CE Marking

9.1 Directives

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

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.

10. Getting Started with CAN Peripherals

A summary of the CAN Peripheral supported MINT keywords can be found in Section 5.2.

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.

10.1 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 10.2 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*

10.2 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 10.3.5).

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 43.

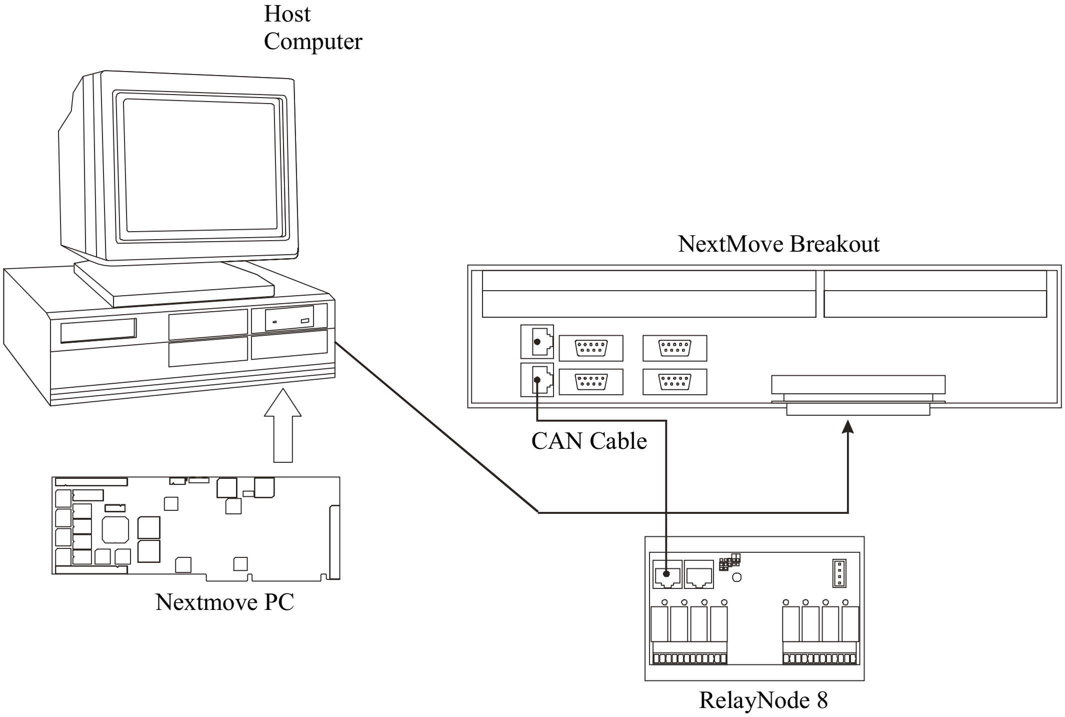


Figure 43: ***NextMove PC*** connected to one *RelayNode 8*.

10.2.1 Jumper setings

1. Make sure that jumpers JP1 and JP2 on the *RelayNode8* are fitted in position 1 (CAN1), as shown in Figure 44.
2. Make sure that jumper JP3 on the *RelayNode8* is fitted, as shown in Figure 44.
3. Make sure that jumpers JP4 and JP5 on the *RelayNode* are not fitted. Refer to Figure 44.

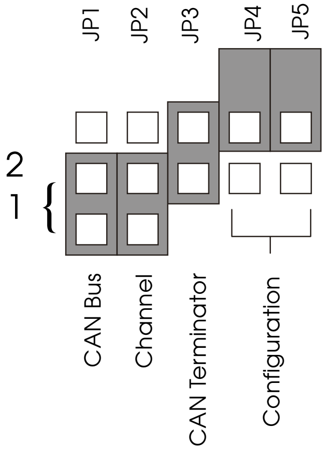


Figure 44: Jumper Settings

10.2.2 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
```

10.3 NextMove PC and CAN Peripherals

10.3.1 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 44.

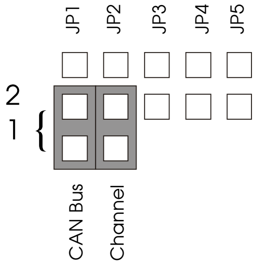


Figure 45: CAN Channel Jumper selection

[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.

10.3.2 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 8.5. 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.

10.3.3 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 46.

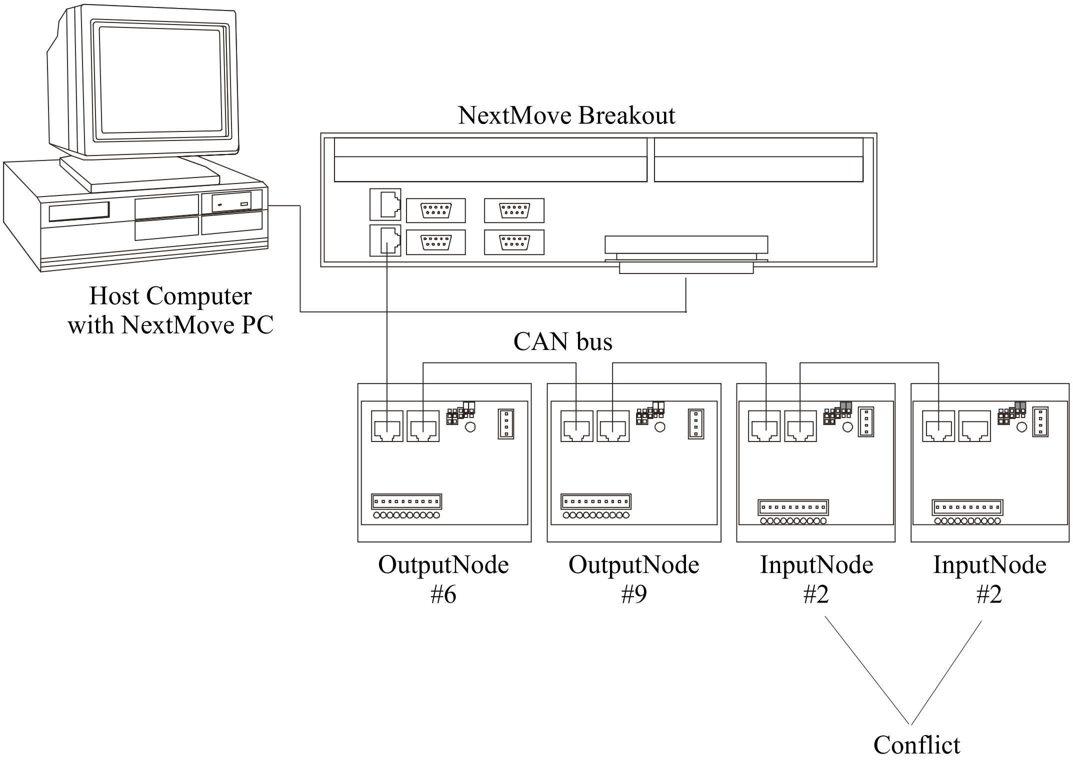


Figure 46: Network with conflicting Node IDs

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 47.

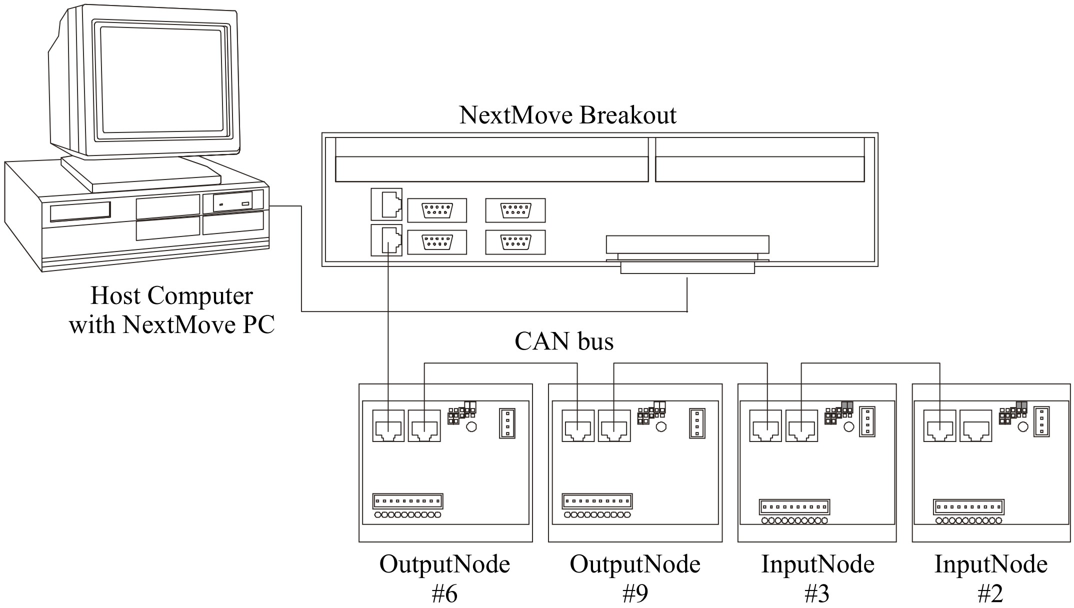


Figure 47: Typical network

To set a Node ID, the node must be statically configured as described in Section 10.3.5.

10.3.4 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 48.

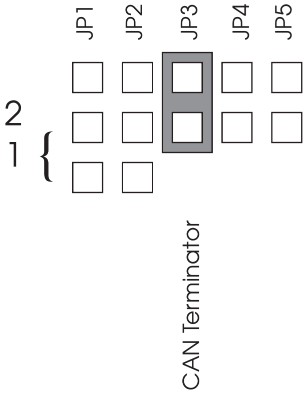


Figure 48: Network Termination Jumper Settings

10.3.5 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*.

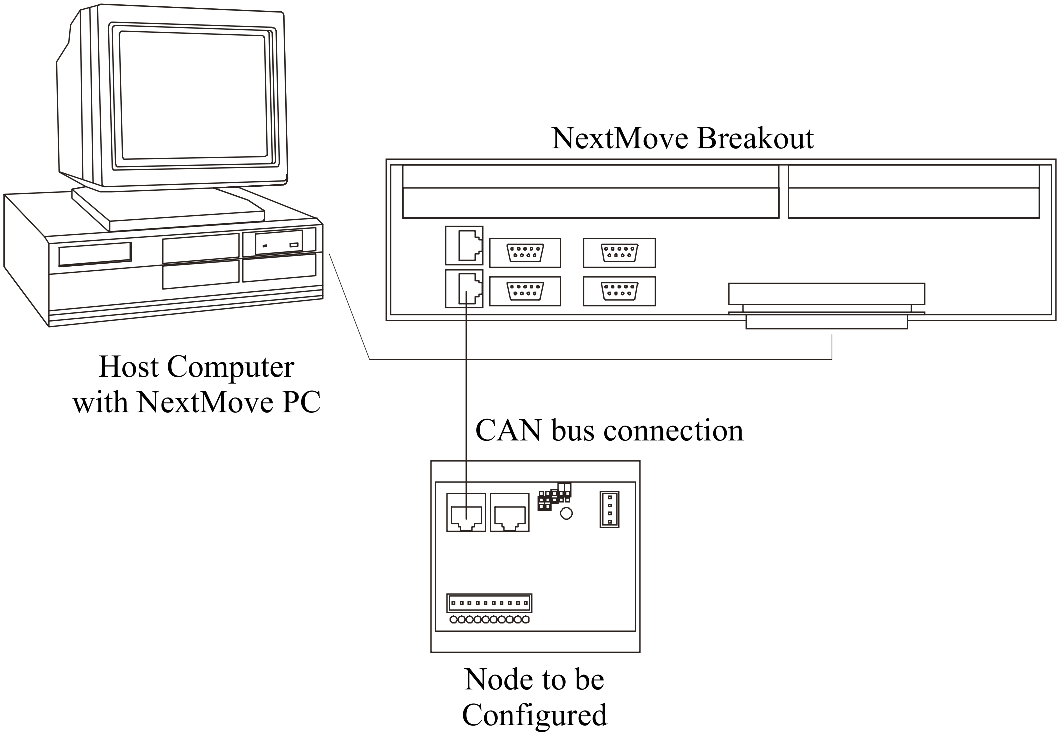


Figure 49: ***NextMove PC*** - Node Configuration

To statically configure a CAN Peripheral:

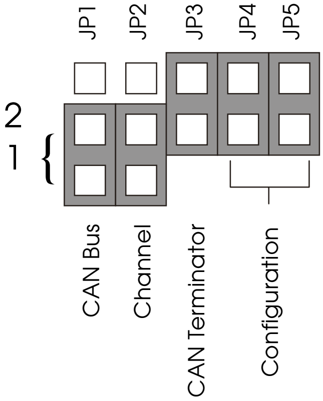


Figure 50: Static Configuration Jumper Position

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 50.

2. Make sure that jumper JP3 (CAN terminator) on the CAN Peripheral is fitted, as shown in Figure 50.
3. Make sure that jumpers JP4 and JP5 (configuration) on the CAN Peripheral are fitted, as shown in Figure 50.
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.

10.3.6 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*.

10.4 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

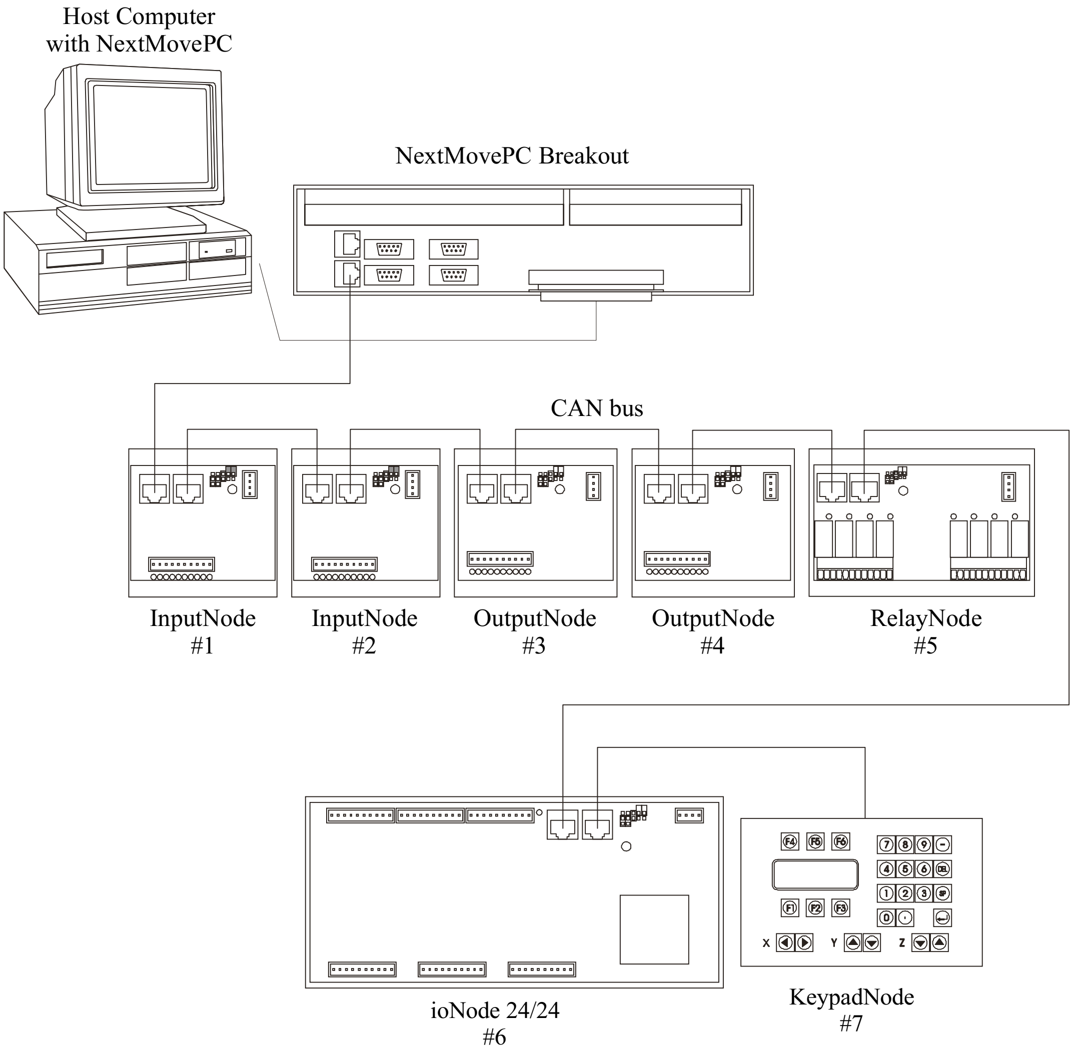


Figure 51: Example multi-node network

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 10.3.5).
2. Statically configure the two *OutputNode 8* nodes with Node IDs 3 and 4 (refer to Section 10.3.5).

3. Statically configure the *RelayNode 8* node with Node ID 5 (refer to Section 10.3.5).
4. Statically configure the *IoNode 24/24* node with Node ID 6 (refer to Section 10.3.5).
5. Statically configure the *KeypadNode* node with Node ID 7 (refer to Section 10.3.5).
6. Terminate the network:

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

   A If the *KeypadNode* is at the end of the network its terminator should be selected by fitting a jumper to JP3.

   A 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.

10.5 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
```

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.

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

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*.

10.6 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

[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.

Keyword	Description
Configuration Keywords	
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

11. MINT Programmer's Toolkit

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`**.

11.1 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:

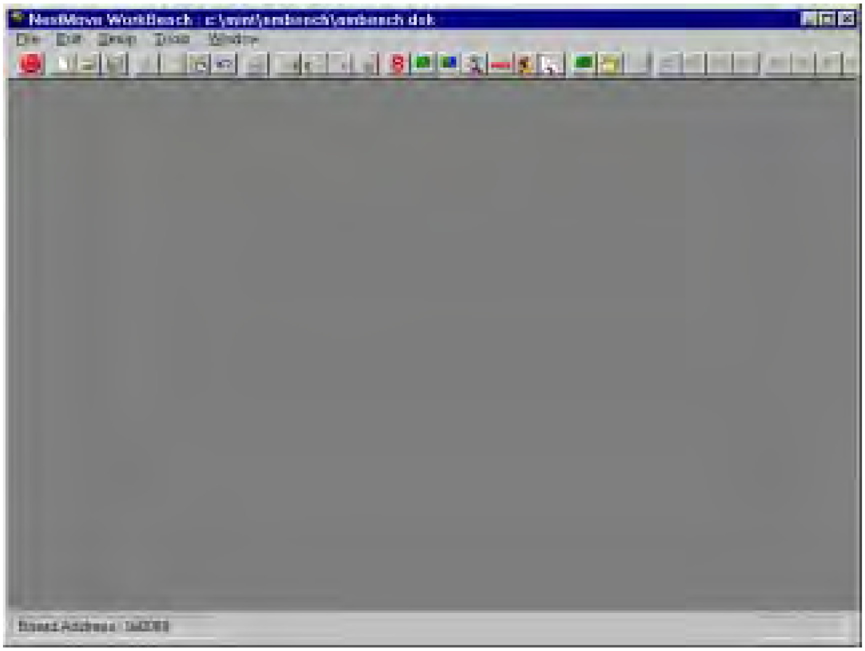


Figure 52: Default *WorkBench* Window

11.2 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.

M1 **to** M10 Execute macros 1 through to 10.

11.3 Selecting the Board Address

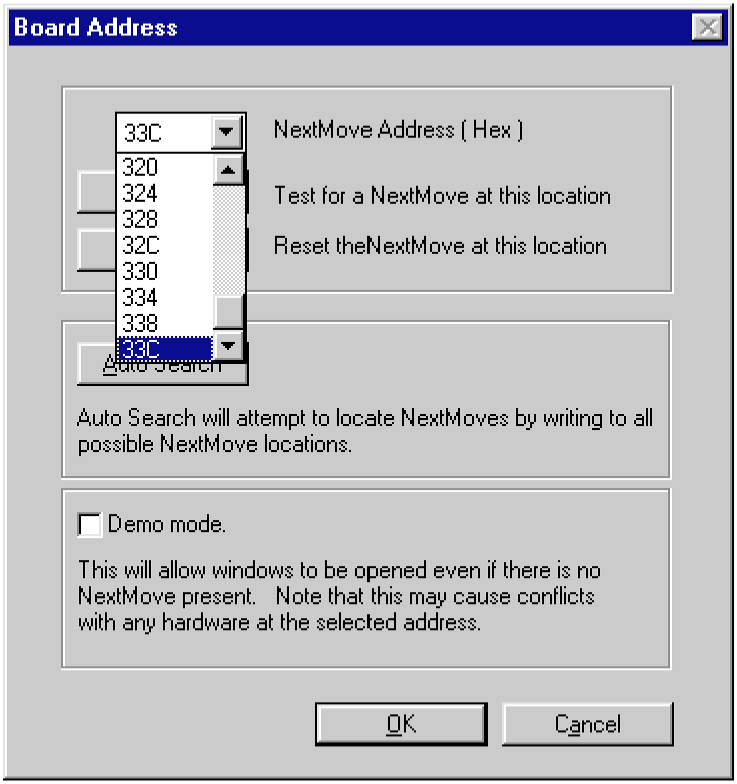
The *WorkBench* only supports one *NextMove PC* at a time although multiple instances of the *WorkBench* can be run. The address of the *NextMove PC* must be supplied before any of the tools can be used. (The *NextMove PC* address is selectable using jumpers - refer to Section 6.1 of this manual for 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 53.



Board Address

NextMove Address (Hex): 33C

Test for a NextMove at this location

Reset theNextMove at this location

Auto Search will attempt to locate NextMoves by writing to all possible NextMove locations.

□ Demo mode.

This will allow windows to be opened even if there is no NextMove present. Note that this may cause conflicts with any hardware at the selected address.

Figure 53: 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.

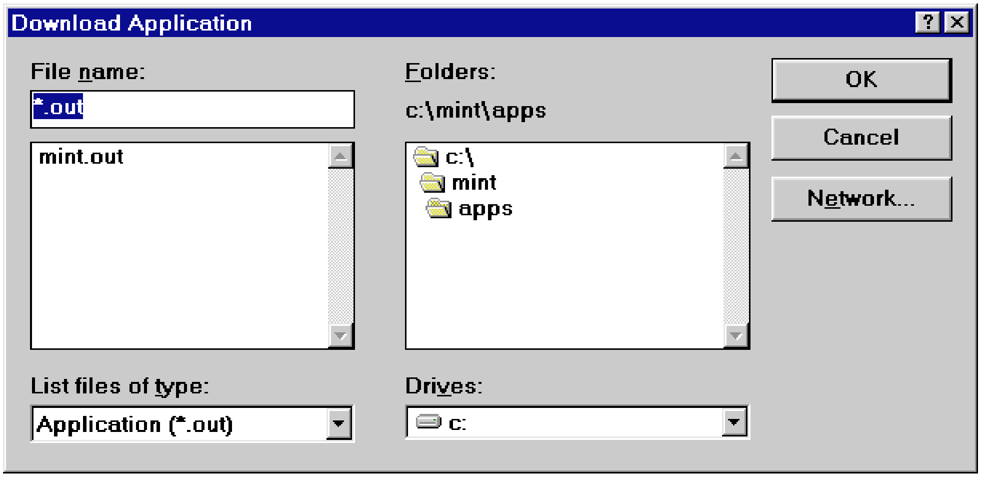
Once an address has been selected, the *WorkBench* will enable any menu items supported by the board at that address. If no application is currently running on *NextMove PC*, only the menu option *Download Application* in the *File* menu will be highlighted. It will be necessary to download an application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

11.4 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.



File name: *.out

Folders: c:\mint\apps

List files of type: Application (*.out)

Drives: c:

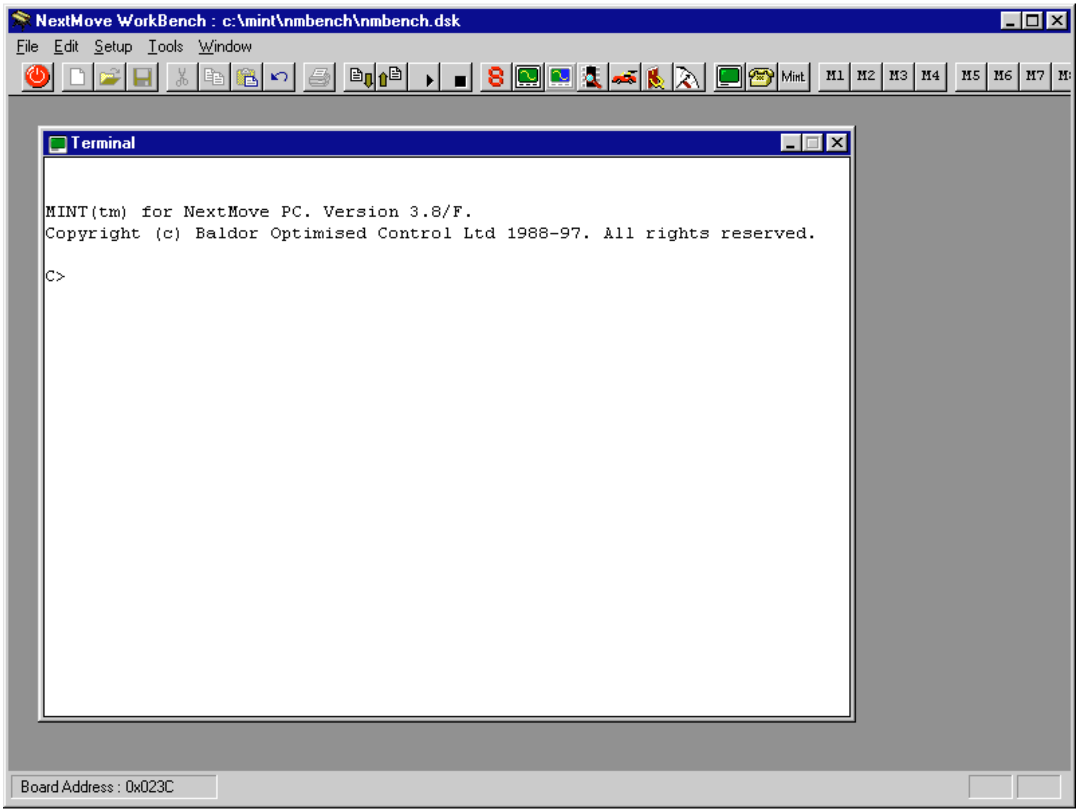
Figure 54: Download Application

11.5 Terminal

The Terminal window provides access to the MINT command line.

To enable the Terminal Window:

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


NextMove WorkBench : c:\mint\nmbench\nmbench.dsk
File Edit Setup Tools Window
Mint
M1 M2 M3 M4 M5 M6 M7
Terminal

```
MINT(tm) for NextMove PC. Version 3.8/F.
Copyright (c) Baldor Optimised Control Ltd 1988-97. All rights reserved.

C>
```

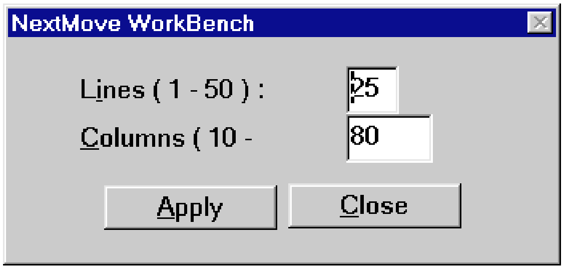
Board Address : 0x023C

Figure 55: 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 56.



Lines (1 - 50) : 25

Columns (10 - 80

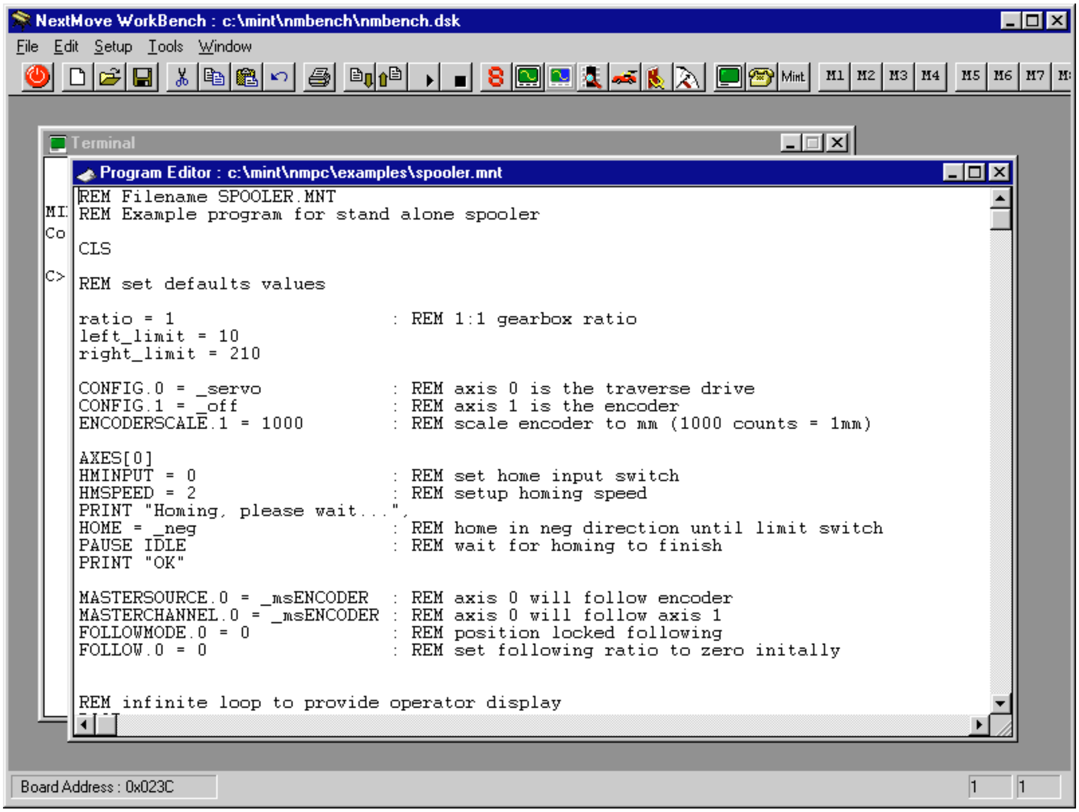
Figure 56: Select Terminal Size Dialog Box

11.6 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 .


NextMove WorkBench : c:\mint\nmbench\nmbench.dsk
File
Edit
Setup
Tools
Window
Mint
M1
M2
M3
M4
M5
M6
M7
Terminal
MI
Co
C>
Program Editor : c:\mint\nmpc\examples\spooler.mnt
Board Address : 0x023C
1
1

```
REM Filename SPOOLER.MNT
REM Example program for stand alone spooler

CLS

REM set defaults values

ratio = 1                     : REM 1:1 gearbox ratio
left_limit = 10
right_limit = 210

CONFIG.0 = _servo             : REM axis 0 is the traverse drive
CONFIG.1 = _off               : REM axis 1 is the encoder
ENCODERSCALE.1 = 1000         : REM scale encoder to mm (1000 counts = 1mm)

AXES[0]
HMINPUT = 0                   : REM set home input switch
HMSPEED = 2                   : REM setup homing speed
PRINT "Homing, please wait...",
HOME = _neg                   : REM home in neg direction until limit switch
PAUSE IDLE                    : REM wait for homing to finish
PRINT "OK"

MASTERSOURCE.0 = _msENCODER   : REM axis 0 will follow encoder
MASTERCHANNEL.0 = _msENCODER  : REM axis 0 will follow axis 1
FOLLOWMODE.0 = 0              : REM position locked following
FOLLOW.0 = 0                  : REM set following ratio to zero initally


REM infinite loop to provide operator display
```

Figure 57: 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 11.7.
- 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.

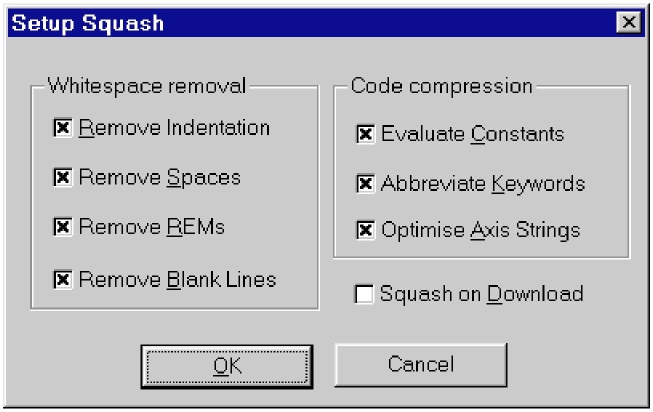
11.7 Squash

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.*



Setup Squash

Whitespace removal

☒ Remove Indentation

☒ Remove Spaces

☒ Remove REMs

☒ Remove Blank Lines

Code compression

☒ Evaluate Constants

☒ Abbreviate Keywords

☒ Optimise Axis Strings

☐ Squash on Download

Figure 58: 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

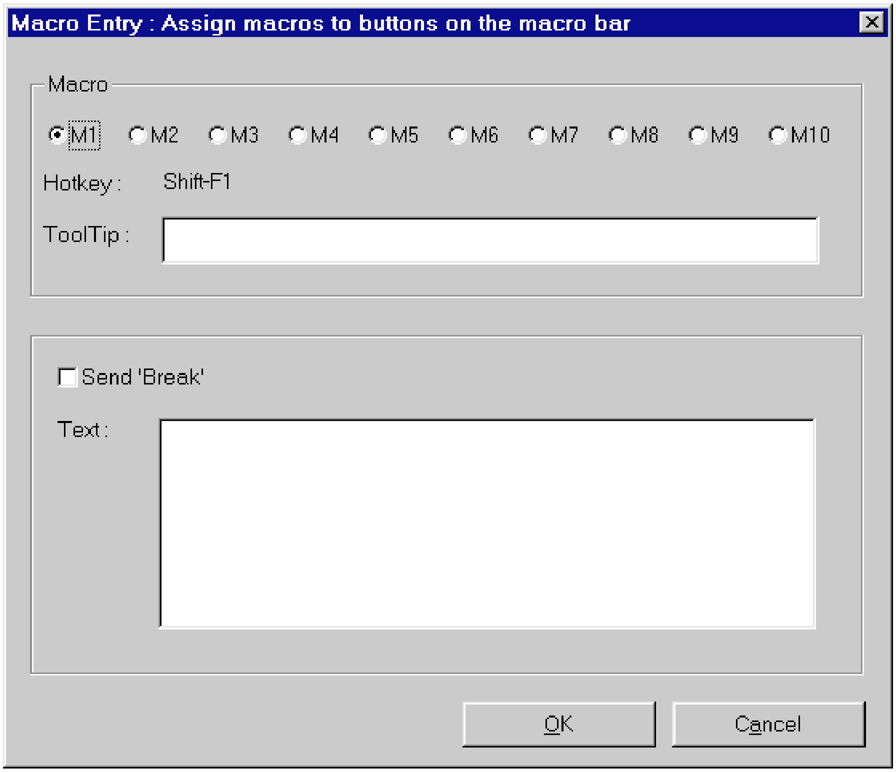
- **Squash on Download** – Clicking this check box Squashes each file as it is downloaded. This allows the file to be edited in the editor in its full 'readable' form, but will download a compact version of the code.

11.8 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*



● M1 ○ M2 ○ M3 ○ M4 ○ M5 ○ M6 ○ M7 ○ M8 ○ M9 ○ M10

Hotkey : Shift-F1

ToolTip :

□ Send 'Break'

Text :

Figure 59: Macro Entry Window

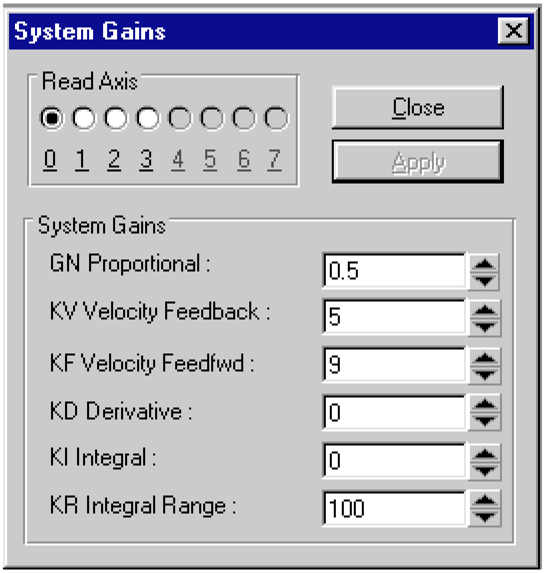
The *Macro Entry* dialog box allows selection of:

- **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.

11.9 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*.



Read Axis

●	○	○	○	○	○	○	○
0	1	2	3	4	5	6	7

System Gains

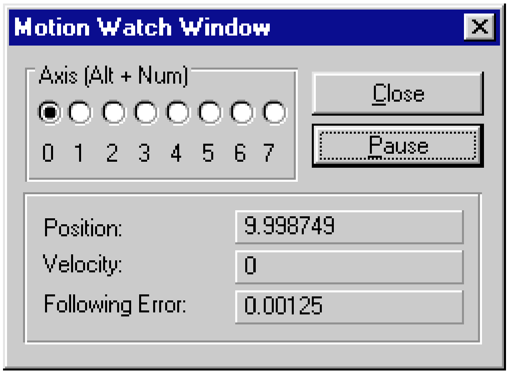
GN Proportional :	0.5
KV Velocity Feedback :	5
KF Velocity Feedfwd :	9
KD Derivative :	0
KI Integral :	0
KR Integral Range :	100

Figure 60: 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.

11.10 Motion WatchWindow

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



Axis (Alt + Num)
● 0 ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7

Position: 9.998749
Velocity: 0
Following Error: 0.00125

Figure 61: 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.

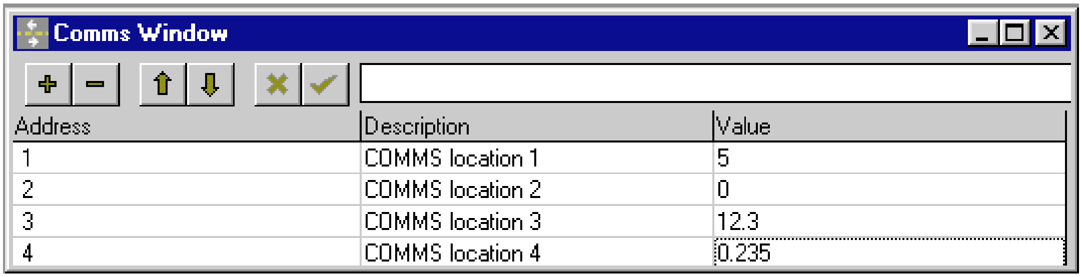
Make a note of which direction gives an increase in position. Repeat for other axes.

11.11 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.



Address	Description	Value
1	COMMS location 1	5
2	COMMS location 2	0
3	COMMS location 3	12.3
4	COMMS location 4	0.235

Figure 62: 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.

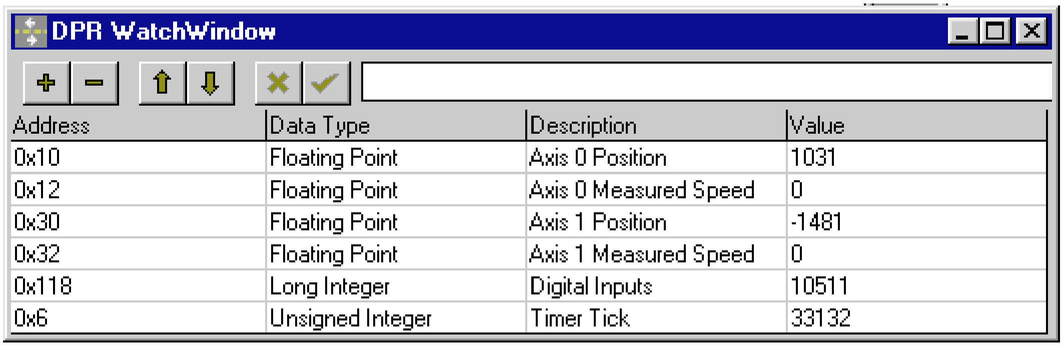
Notes:

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.

11.12 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 63: 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*.

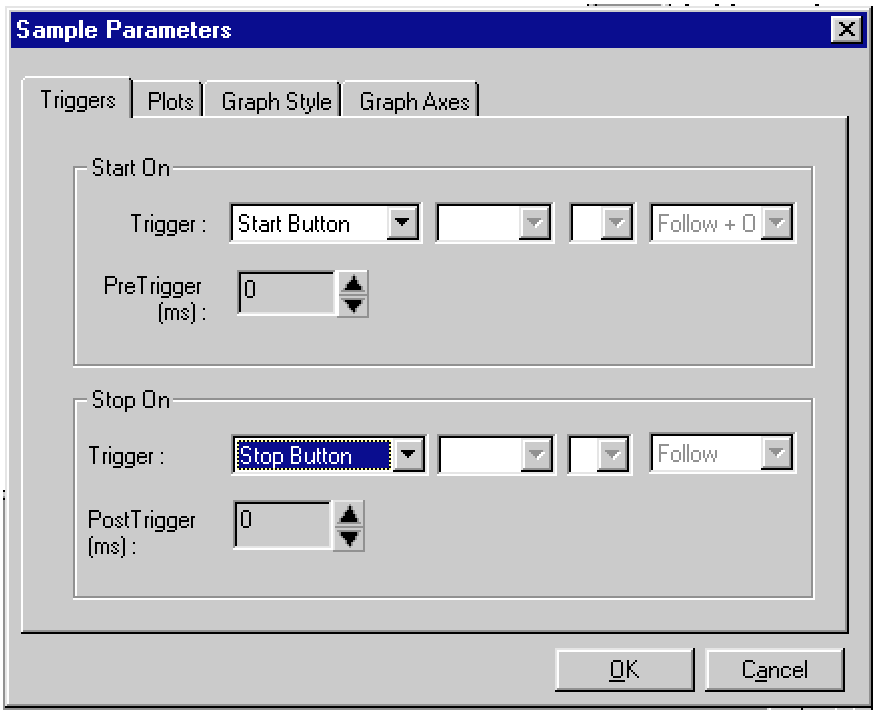
11.13 Software Oscilloscope

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.



Start On

Trigger : Start Button | Follow + 0

PreTrigger (ms) : 0

Stop On

Trigger : Stop Button | Follow

PostTrigger (ms) : 0

Figure 64: 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.

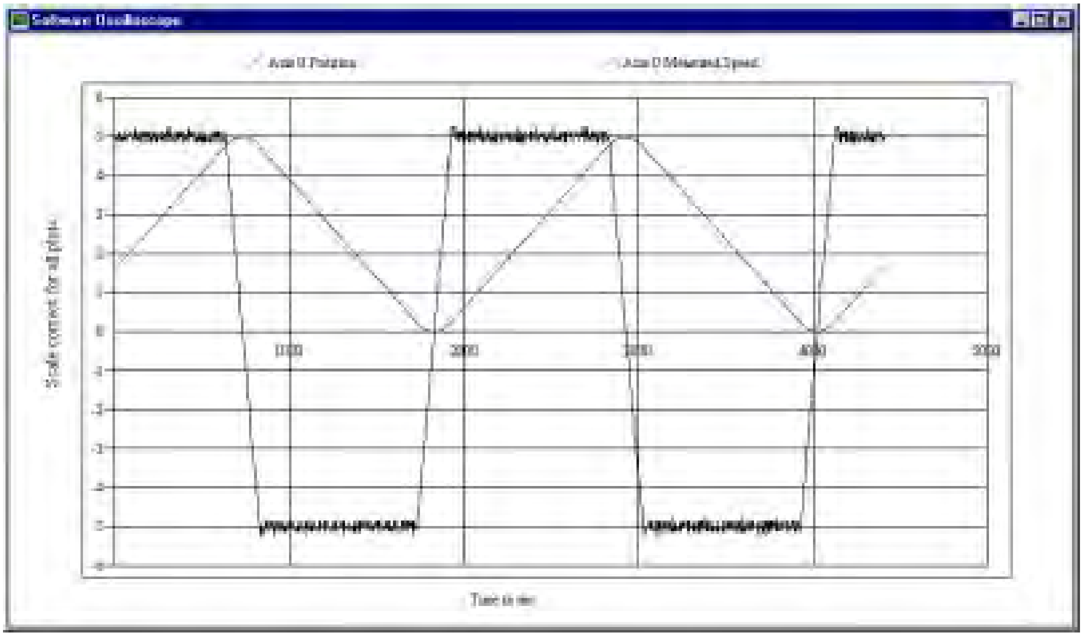


Figure 65: Software Oscilloscope

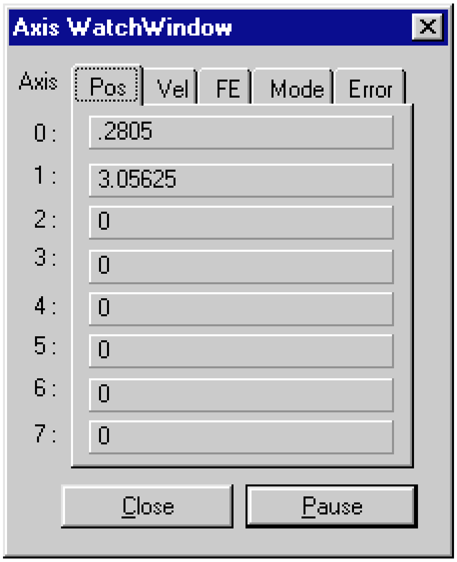
Once the graph has been plotted the *Graph Style* can be changed by clicking on the icon from the toolbar. This opens the *Sample Parameters* dialogue box but only the *Graph Style* tab is enabled.

11.14 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 8 on the toolbar.



Axis	Pos
0 :	.2805
1 :	3.05625
2 :	0
3 :	0
4 :	0
5 :	0
6 :	0
7 :	0

Figure 66: Axis WatchWindow

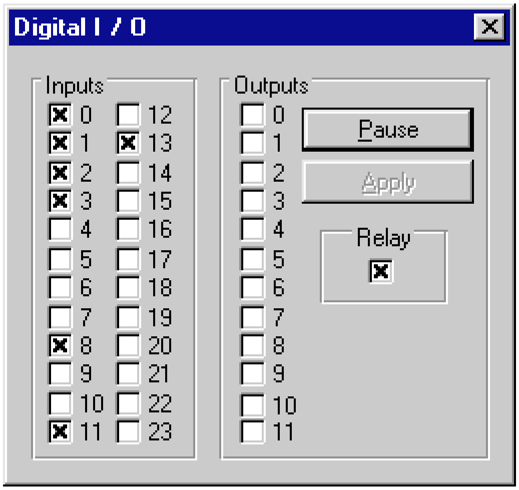
Double clicking on an axis value will display the *Motion WatchWindow* for that axis.

11.15 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.



Digital I / O

Inputs

☒ 0 □ 12
☒ 1 ☒ 13
☒ 2 □ 14
☒ 3 □ 15
□ 4 □ 16
□ 5 □ 17
□ 6 □ 18
□ 7 □ 19
☒ 8 □ 20
□ 9 □ 21
□ 10 □ 22
☒ 11 □ 23

Outputs

□ 0
□ 1
□ 2
□ 3
□ 4
□ 5
□ 6
□ 7
□ 8
□ 9
□ 10
□ 11

Relay ☒

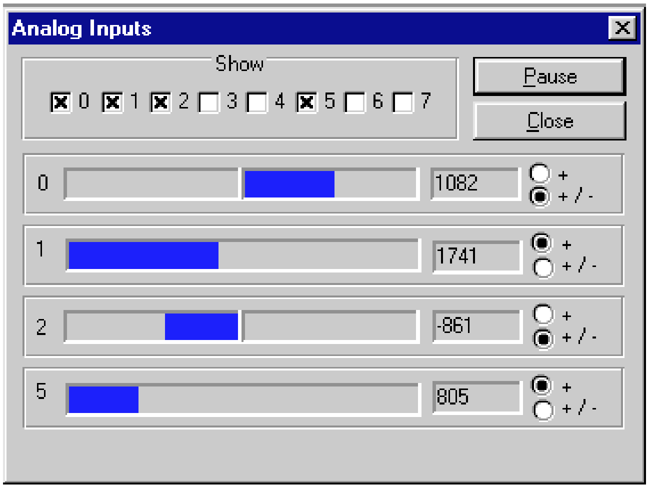
Figure 67: Digital I/O

11.16 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.



Analog Inputs

Show

☒ 0 ☒ 1 ☒ 2 □ 3 □ 4 ☒ 5 □ 6 □ 7

0 1082 ○ + ● + / -

1 1741 ● + ○ + / -

2 -861 ○ + ● + / -

5 805 ● + ○ + / -

Figure 68: Analog Inputs

11.17 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.*

11.18 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.

12. Trouble Shooting Guide

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 **`SO[0,1,2,3]`** 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.

13. Breakout Board

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 69. 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.

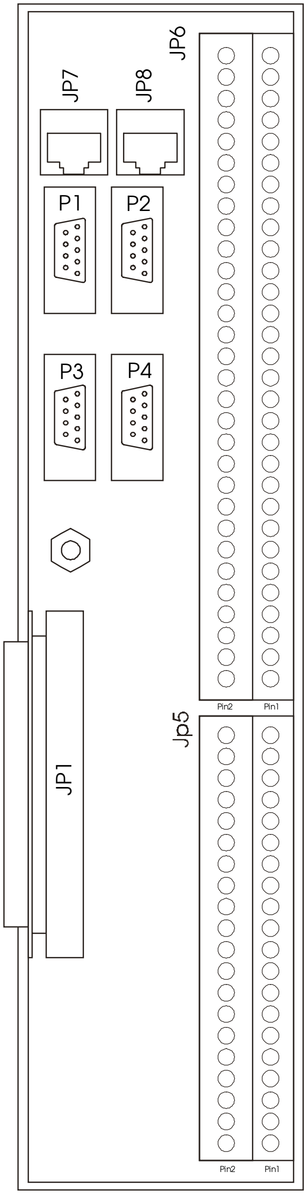
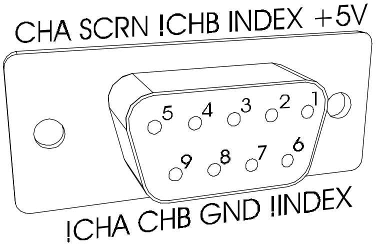


Figure 69: Breakout Board

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

13.1 Encoders: P1, P2, P3 and P4

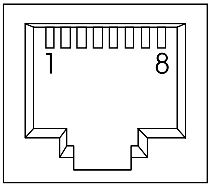
The encoder inputs are brought out onto 9 pin D-type female sockets. The shell of the connector is connected to chassis (screen/shield).



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

13.2 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

13.3 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*

Pin	Signal	Function
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 refereed to USR-V+ and USR-GND. All other pins are at the potential of the host.

13.4 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

Pin	Signal	Function
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
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

Pin	Signal	Function
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 refereed to USR-V+ and USR-GND. All other pins are at the potential of the host.

The relay pins are fully floating.

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.