USER'S MANUAL

NPMC5023-2104

NPM Nippon Pulse Motor Co., Ltd.

First edition, August 2000

Hardware Information Revision: 1.53
Software Information Revision: 2.0

Table of Contents

1. Introduction to NPMC5023-2104

NPMC5023-2104 is a PC104 BUS 2 axis stepper controller card. Utilizing the most powerful stepper pulse generation chip PCL5023, NPMC5023-2104 controller board is packed with advanced features found only in high-end stepper controller boards:

- 2 axis stepper pulse/direction control
- Maximum output Frequency of 4.9M pulse per second
- Linear and S-Curve Acceleration/Deceleration
- Encoder input for position verification with +/- 134M pulse range
- Opto-isolated inputs for +/- Limits, Home, and Alarm
- Driver enable output
- Interrupt capability for error and move finish

NPMC5023-2104 can be plugged into any PC104 control system. If the control system has Windows 3.1, 95, 98 operating system, getting the controller up and running can be done in a few minutes using the user friendly graphical interface. You can quickly check all the available functions of the controller. The graphical user interface program also has a programming environment which you can use to cycle motion routines.

Writing an application specific program is an easy task with all the driver functions provided in C source code.

2. Getting Started

NPMC5023-2102 requires PC104 BUS. The controller is shipped with the following default jumper settings:

Controller Base Address:	0x200 (hex)
Interrupt Number:	Disabled
Limit Switch Polarity:	Normally Open

If this is the first time using the board, you can quickly test the board by installing the card and running the test program.

To install the stepper card:

1) Turn off the power to the system.
2) Select the jumper setting and attach the cables to the card.
3) Plug the card to the PC104 BUS.
4) Turn on the power to the PC104 setup.

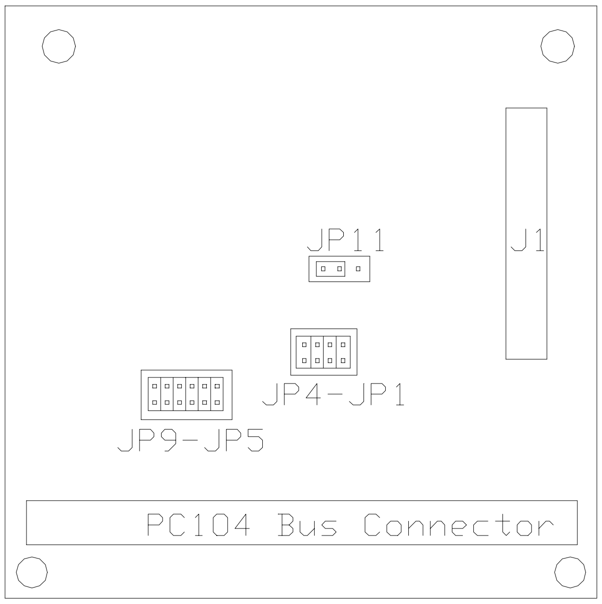
To install the Software:

1) The floppy that comes with the controller contains the install program. Go to Windows and run the setup program from the floppy.
2) All the programs and files will be copied to the specified directory.

To run the Software:

Double click on the stepper motor icon to start the test program. For detailed information, see page 10.

3. Connector and Jumper Location



J1	-	30 pin IDC connector for stepper control
JP1-JP4	-	Bus address select jumper
JP5-JP9	-	Interrupt select jumper
JP11	-	Limit switch polarity select jumper for normally open or normally closed limit sensor configuration.

(Home and Alarm sensor polarity can be set by software)

4. **Connector Information**

4.1 Connector: J1- Stepper Motor Control

Description	Pin		Description
+5	1	2	GND
Pulse 1	3	4	Dir 1
Enable 1	5	6	Encoder Z 1
Encoder A1	7	8	Encoder B 1
Home 1	9	10	+Lim1
-Lim 1	11	12	Alarm 1
+5V	13	14	GND
Pulse 2	15	16	Dir 2
Enable 2	17	18	Encoder Z2
Encoder A2	19	20	Encoder B2
Home 2	21	22	+Lim 2
-Lim 2	23	24	Alarm2
Vss	25	26	Vss
NC	27	28	NC
NC	29	30	NC
NC	31	32	NC
STP	33	34	STA

NOTE: For detailed information on STA and STP, please refer to NPMC5023 chip manual.

5. Jumper Information

5.1 Jumpers JP4 to JP1 – Bus Address Selector

JP4	JP3	JP2	JP1	Base Address
				0x3E0 (hex)
X				0x3C0 (hex)
	X			0x3A0 (hex)
X	X			0x380 (hex)
		X		0x360 (hex)
X		X		0x340 (hex)
	X	X		0x320 (hex)
X	X	X		0x300 (hex)
			X	0x2E0 (hex)
X			X	0x2C0 (hex)
	X		X	0x2A0 (hex)
X	X		X	0x280 (hex)
		X	X	0x260 (hex)
X		X	X	0x240 (hex)
	X	X	X	0x220 (hex)
X	X	X	X	0x200 (hex)

5.2 Jumpers JP5-JP9 – Interrupt Number Selector

JP5	JP6	JP7	JP8	JP9	Interrupt #
X					IRQ 7
	X				IRQ 6
		X			IRQ 5
			X		IRQ 4
				X	IRQ 3

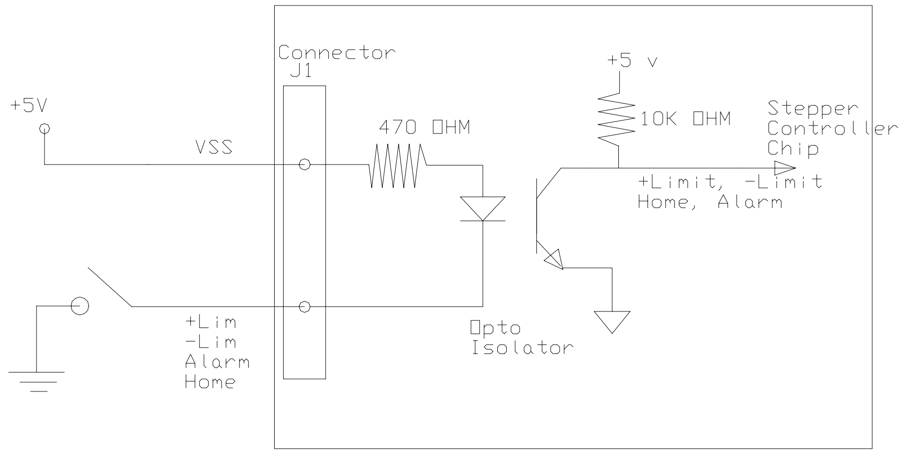
5.3 Jumper JP11 – Limit input polarity selector

JP11	Description
LEFT	For Normally Closed Switches
RIGHT	For Normally Open Switches

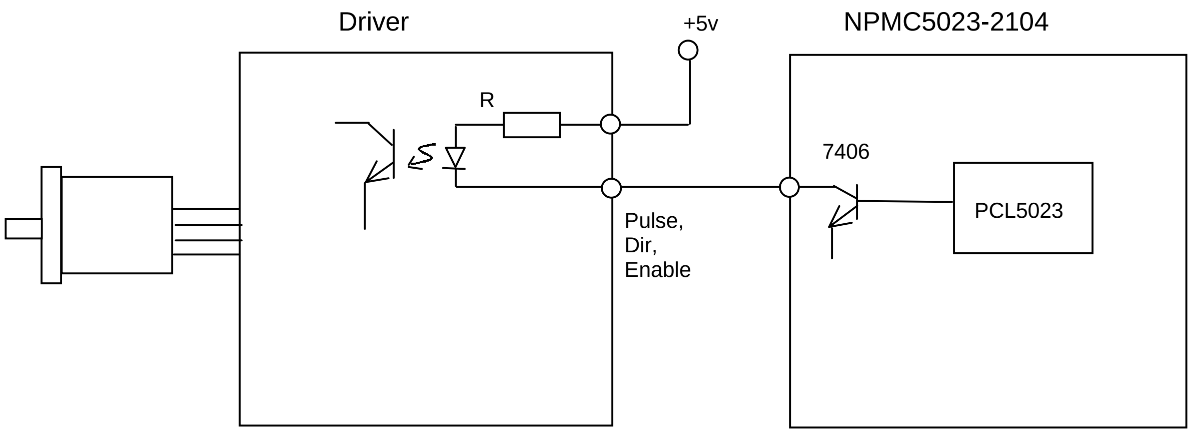
X	*– jumper installed*
LEFT	*– Left two pins are jumped*
RIGHT	*– Right two pins are jumped*

6. Electrical Information

6.1 + Limit, -Limit, Home, and Alarm Inputs:



6.2 Pulse, Direction, Enable Outputs



7. Introduction to NPMC Software

NPMC-S is user friendly graphical program for quickly testing all the features of the following stepper controller cards:

NPMC-2ISA– ISA BUS 2 axis + 48 DIO
NPMC-4ISA– ISA BUS 4 axis + 48 DIO
NPMC-2104– PC104 BUS 2 axis stepper

Program can be run in Windows 3.1, 95, and 98.

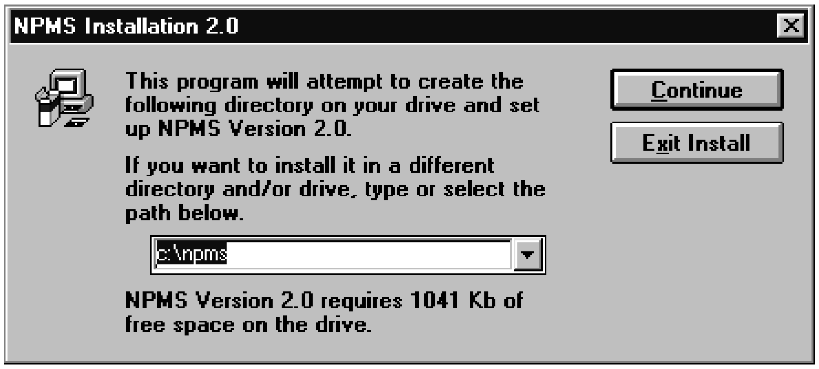
Once the testing is done, you can quickly move on to your application using the C source code drivers.

8. Installing NPMC-S

1. Insert the NPMC-S installation disk into the floppy drive.

2. Run or double click the install.exe program in the floppy disk:

3. The installation will show the following dialog box:


NPMS Installation 2.0

This program will attempt to create the following directory on your drive and set up NPMS Version 2.0.

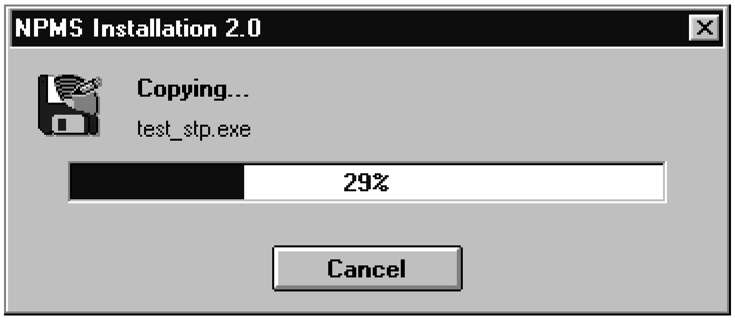
If you want to install it in a different directory and/or drive, type or select the path below.

c:\npms

NPMS Version 2.0 requires 1041 Kb of free space on the drive.

4. Select the default directory ***C:\NPMCS*** and click on ***Continue*** button.

5. Following dialog box will pop up to indicate that the installation is progressing:



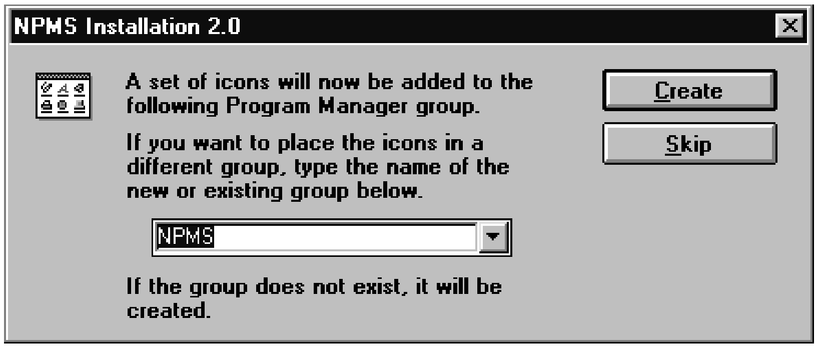
NPMS Installation 2.0

Copying...

test_stp.exe

29%

6. After successful copy to the directory following dialog box will pop up:



NPMS Installation 2.0

A set of icons will now be added to the following Program Manager group.

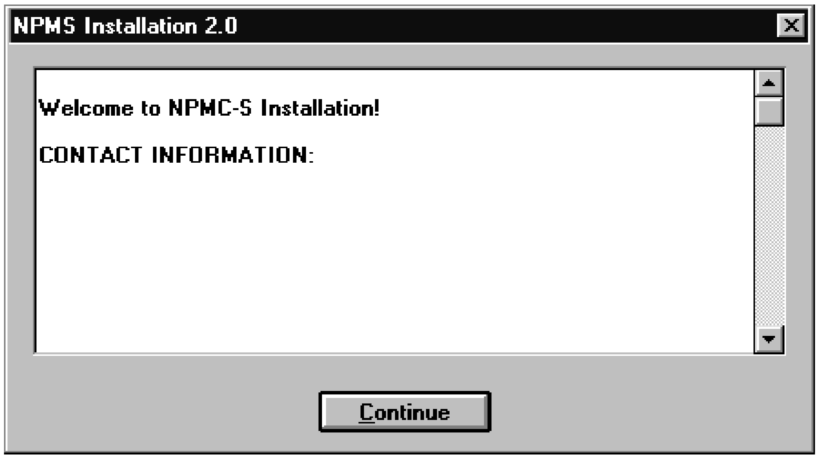
If you want to place the icons in a different group, type the name of the new or existing group below.

NPMS

If the group does not exist, it will be created.

7. Select the program group to be added to your Windows selection. Click on ***Create*** button.

8. After creating the program group following dialog box will pop up:

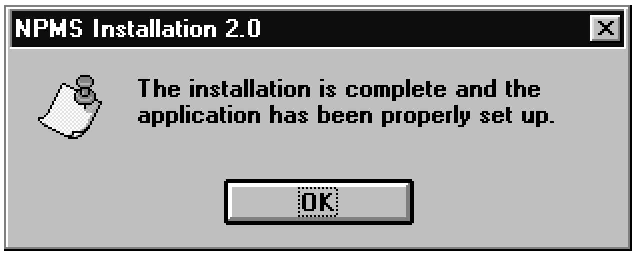

NPMS Installation 2.0

Welcome to NPMC-S Installation!

CONTACT INFORMATION:

9. Any last minute information will be included here. Click ***Continue*** after reading the information.

10. Final dialog box will show to indicate that the installation was successful.

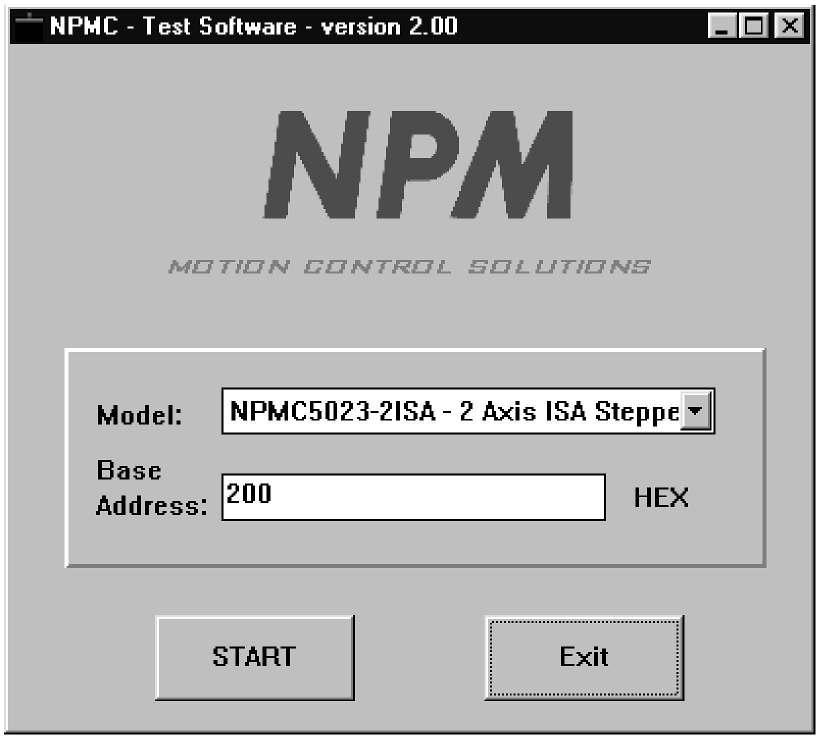

NPMS Installation 2.0

The installation is complete and the application has been properly set up.

9. Running NPMCS

1. From the Windows program manager, click on the following icon:

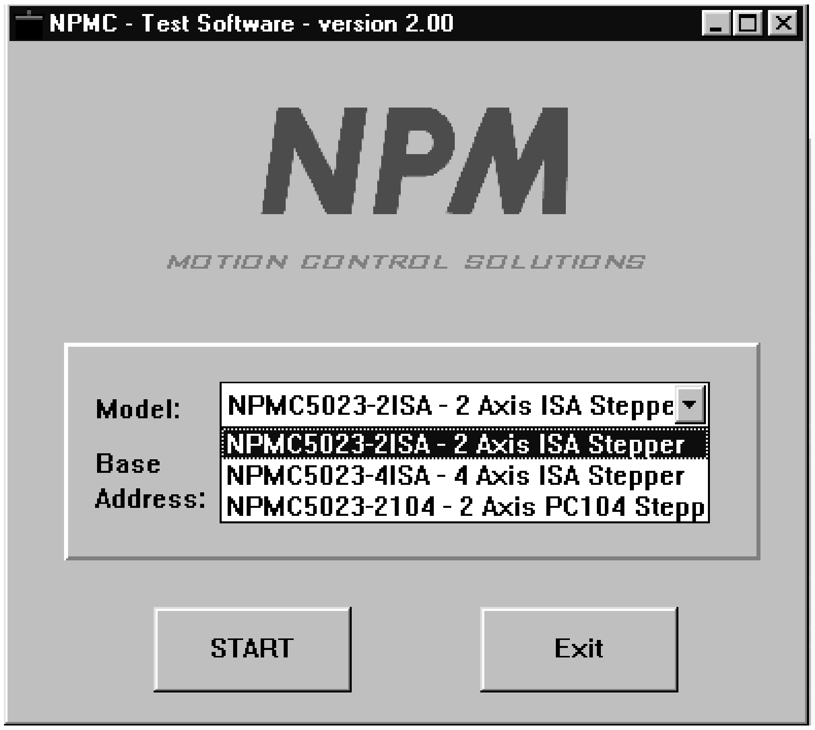
2. Following opening screen will pop up:


NPMC - Test Software - version 2.00

Model: NPMC5023-2ISA - 2 Axis ISA Steppe

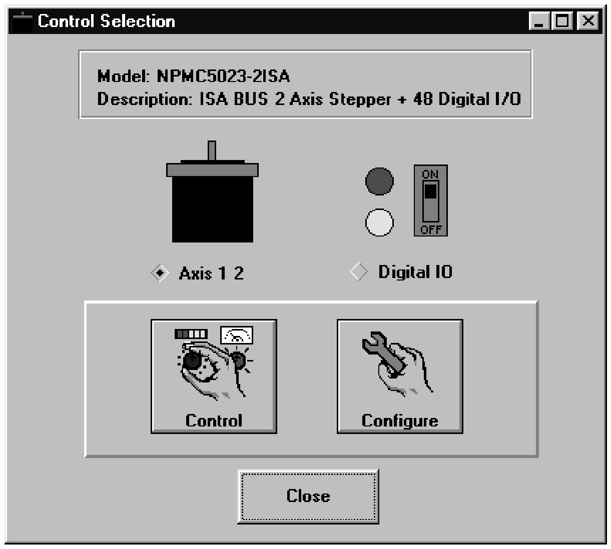
Base Address: 200 HEX

3. From dialog box, select the controller model and enter the base address.


NPMC - Test Software - version 2.00
NPM
MOTION CONTROL SOLUTIONS
Model:
Base Address:
NPMC5023-2ISA - 2 Axis ISA Steppe
NPMC5023-2ISA - 2 Axis ISA Stepper
NPMC5023-4ISA - 4 Axis ISA Stepper
NPMC5023-2104 - 2 Axis PC104 Stepp
START
Exit

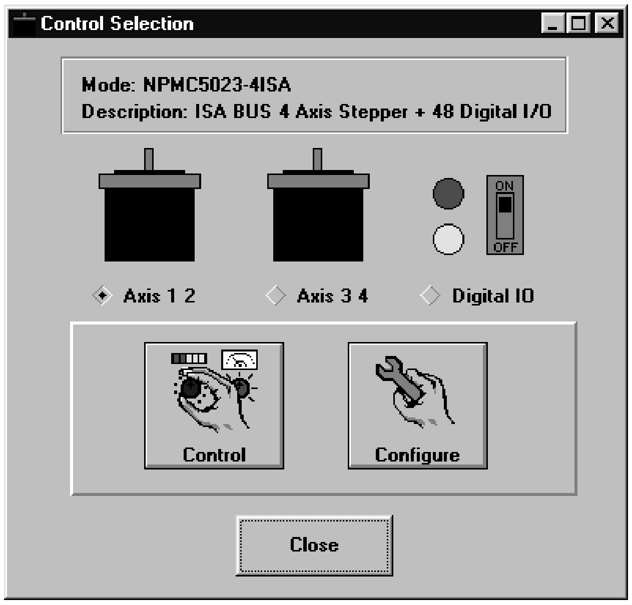
4. After selecting the model number click on the ***Start*** button. Depending on which model you selected one of the following screens will pop up:

For NPMC-2ISA



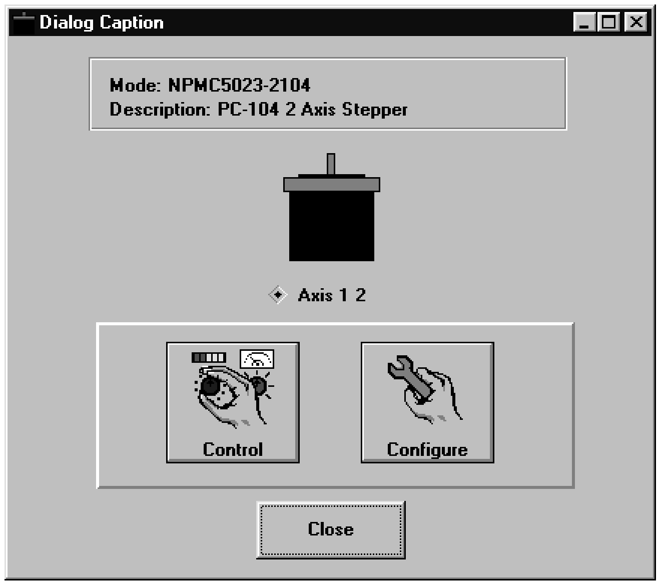
Model: NPMC5023-2ISA
Description: ISA BUS 2 Axis Stepper + 48 Digital I/O

For NPMC-4ISA



Mode: NPMC5023-4ISA
Description: ISA BUS 4 Axis Stepper + 48 Digital I/O

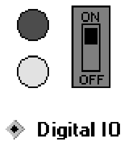
For NPMC-2104

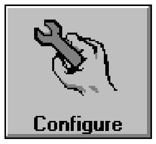

Dialog Caption

Mode: NPMC5023-2104
Description: PC-104 2 Axis Stepper

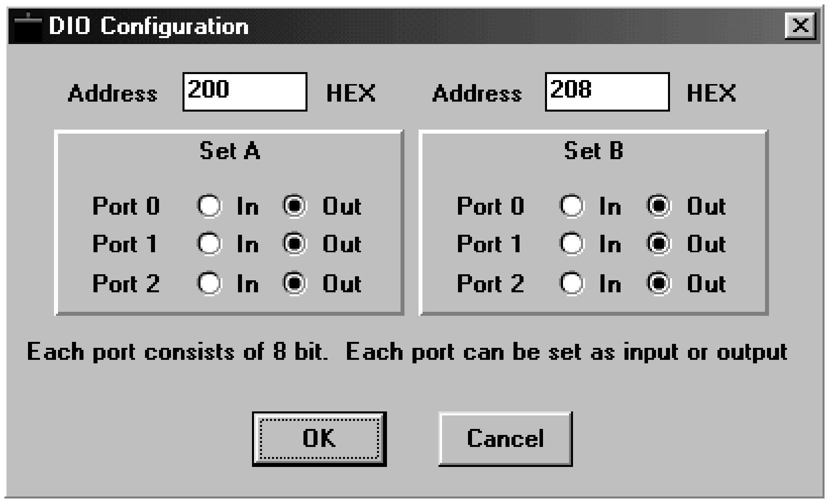
Depending on the model you can control Axis 1 and 2, Axis 3 and 4, and 48 Digital IO’s.

5. For NPMC-2ISA and NPMC-4ISA there are 48 Digital IO's which can be configured as inputs or outputs. To configure the Digital IO's, select the Digital IO radio button and click on configure button:





Following screen will appear:



Address	200	HEX	
Address	208	HEX	

Set A

Port 0	In	[✓ Out]
Port 1	In	[✓ Out]
Port 2	In	[✓ Out]

Set B

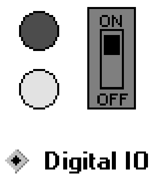
Port 0	In	[✓ Out]
Port 1	In	[✓ Out]
Port 2	In	[✓ Out]

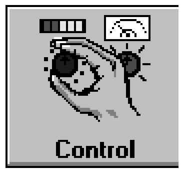
Each port consists of 8 bit. Each port can be set as input or output

You will not be able to change the address of the digital IO's since the base address selection was done on the first screen. The two addresses are shown for your application development.

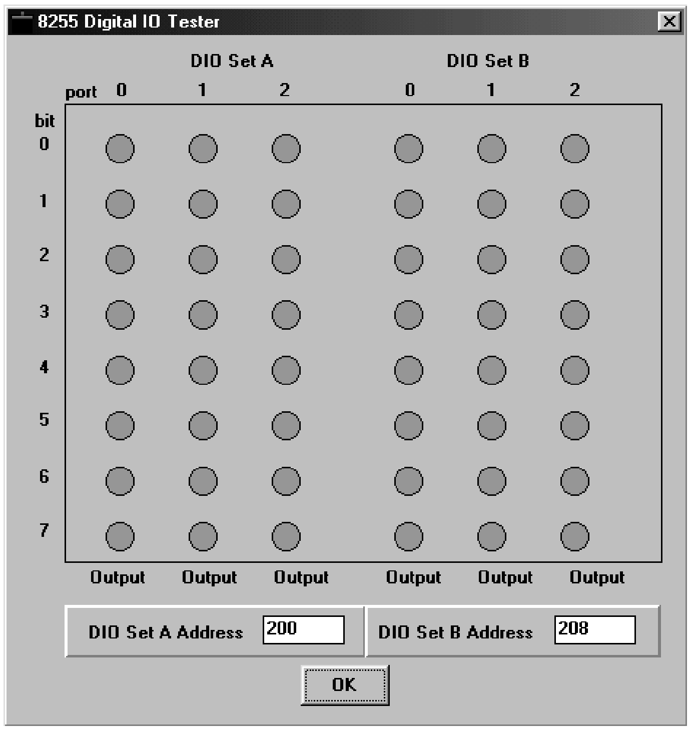
There are two sets of digital IO's: Set A and Set B. Each set contains 3 ports. Each port has 8 bits of digital IO's. Click on the either the in or out configuration or each port. When you are done, click on ***OK*** and you will return to the previous dialog box.

6. To control the Digital IO's, select the digital IO and click on the control button:





Following screen will appear:



DIO Set A Address 200

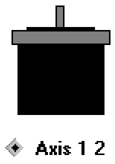
DIO Set B Address 208

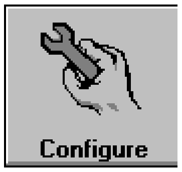
From the DIO control dialog box, you can set and view all the digital IO's.

7. Hit OK button when you are done and you will return to the previous dialog box.

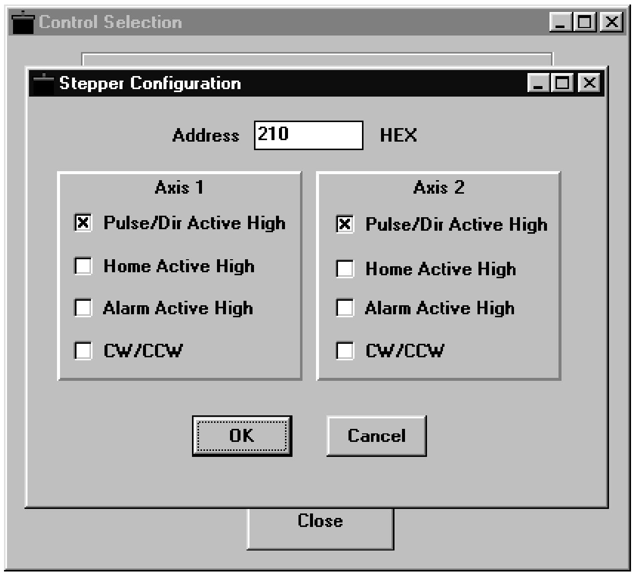
NPM

8. To configure the stepper controller select the Stepper and click on Configure button:





9. Following setup dialog box will appear:



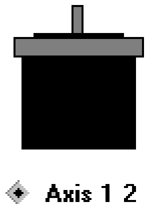
Address 210 HEX

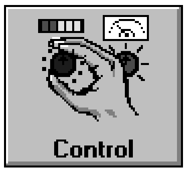
Axis 1
[✓ Pulse/Dir Active High]
Home Active High
Alarm Active High
CW/CCW

Axis 2
[✓ Pulse/Dir Active High]
Home Active High
Alarm Active High
CW/CCW

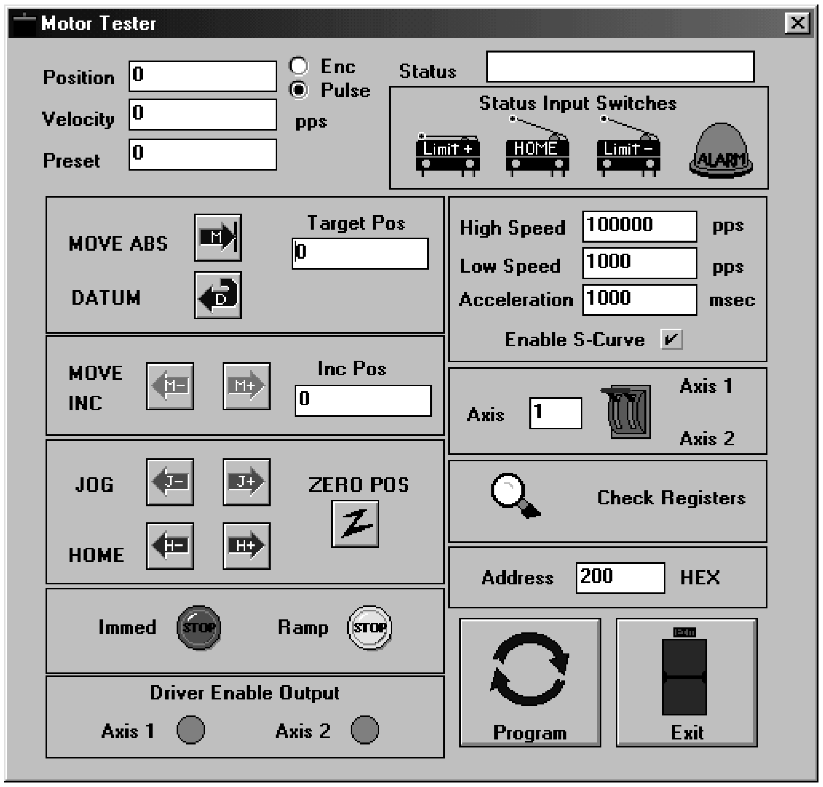
From the stepper configuration, setup the polarity of the pulse/dir output signal, polarity of the Home and Alarm inputs. For Pulse/Dir outputs, two pulse trains can be set by CW/CCW. For the limit switch polarity settings, hardware jumpers are used. Refer to the Hardware manual on limit polarity setting. Click on ***OK*** after configuration.

10. To control the stepper motors select the stepper selection and click on Control button.





11.Following dialog box will appear:



Motor Tester

Position: 0
Velocity: 0 pps
Preset: 0
( ) Enc [✓ Pulse]
Status:

MOVE ABS — Target Pos: 0
DATUM

High Speed: 100000 pps
Low Speed: 1000 pps
Acceleration: 1000 msec
Enable S-Curve: ✓

MOVE INC — Inc Pos: 0

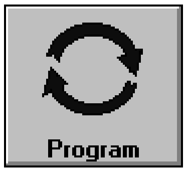
Axis: 1 (Axis 1 / Axis 2)

Address: 200 HEX

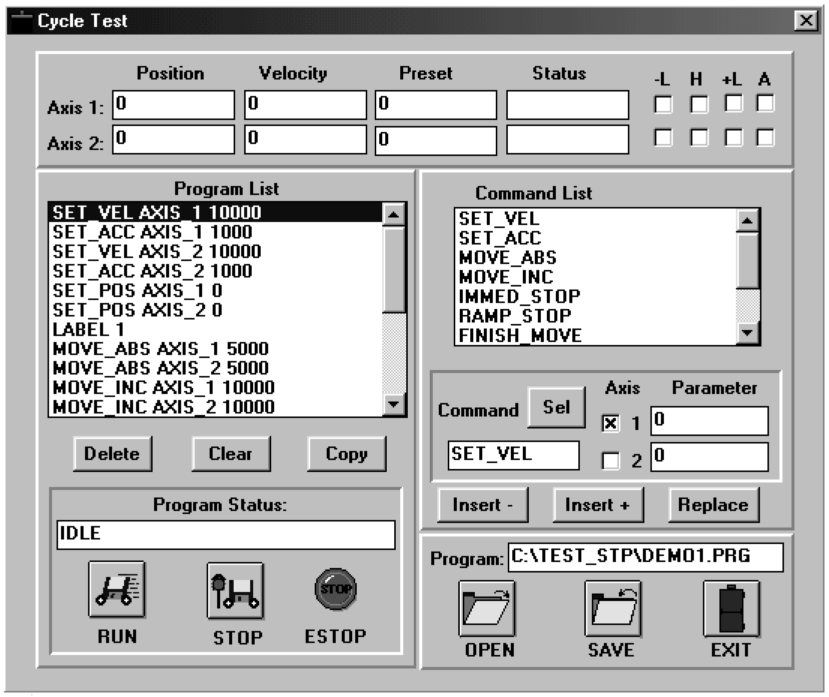
Driver Enable Output
Axis 1 ●   Axis 2 ●

From this screen, you can do full control of the stepper motors. Most of the buttons and descriptions should be intuitive and self-explanatory. The ***Datum*** button means return the motor back to zero position. There is also a ***Check Registers*** button that will allow low level access to the stepper control chip. It is recommended that you do not access the Registers unless you are familiar with the low levels of the stepper chip. Refer to the 5023 chip manual for details.

12.RSTP program has a powerful built in programming environment to test motion sequences. To access this environment click on the ***Program*** button:



13. Following dialog box will appear:



	Position	Velocity	Preset	Status	-L	H	+L	A
Axis 1:	0	0	0		☐	☐	☐	☐
Axis 2:	0	0	0		☐	☐	☐	☐

Program List

```
SET_VEL AXIS_1 10000
SET_ACC AXIS_1 1000
SET_VEL AXIS_2 10000
SET_ACC AXIS_2 1000
SET_POS AXIS_1 0
SET_POS AXIS_2 0
LABEL 1
MOVE_ABS AXIS_1 5000
MOVE_ABS AXIS_2 5000
MOVE_INC AXIS_1 10000
MOVE_INC AXIS_2 10000
```

Command List

```
SET_VEL
SET_ACC
MOVE_ABS
MOVE_INC
IMMED_STOP
RAMP_STOP
FINISH_MOVE
```

Command: SET_VEL

Axis	Parameter
☒ 1	0
☐ 2	0

Program Status: IDLE

Program: C:\TEST_STP\DEMO1.PRG

From the programming environment you can load, create, edit, save, and run simple motion sequences.

10. Writing your Application

To write your own application, use the source code provided in the C language. You can directly access the stepper control functions using C driver source code. You would need to add to your application project and compile and link to your application. Sample C programs using the C driver source is provided in the floppy diskette.

11. Stepper Access Functions

11.1. Move Command Functions

(Note: All motion command speed and accelerations are set by the set_speed_accel() functions)

Move to Absolute Position

int movea (int base, int axis, long pos, long encoder_per_rev, long pulse_per_rev);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1.
pos – target position with range of +/- 134M
encoder_per_rev – encoder resolution per revolution.
    Set to 0 if encoder is not used.
pulse_per_rev – stepper pulse resolution. This is used
    with encoder. If encoder is not used, set it to 0.

Linear Interpolated Move to Absolute Position

int dual_movea(int base, long pos1, long pos2, long encoder_per_rev1, long encoder_per_rev2, long pulse_per_rev1, long pulse_per_rev2)

Arguments:
base – base address of the controller card
pos1 – target position of axis 1
pos2 – target position of axis 2
encoder_per_rev1 – axis 1 encoder resolution.
    Set to 0 if encoder is not used.
encoder_per_rev2 – axis 2 encoder resolution.
    Set to 0 if encoder is not used.
pulse_per_rev1 – axis 1 stepper pulse resolution.
    This is used with encoder. If encoder is not
    used, set it to 0.
pulse_per_rev2 – axis 2stepper pulse resolution.
    This is used with encoder. If encoder is not
    used, set it to 0.

Move to Incremental Position

int movei(int base, int axis, long pos);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1.
pos – relative move amount. Use negative value for
negative direction.

Linear Interpolated Move to Incremental Position

int dual_movei(int base, long pos1, long pos2, long encoder_per_rev1, long encoder_per_rev2, long pulse_per_rev1, long pulse_per_rev2)

Arguments:
base – base address of the controller card
pos1 – incremental position of axis 1
pos2 – incremental position of axis 2
encoder_per_rev1 – axis 1 encoder resolution.
Set to 0 if encoder is not used.
encoder_per_rev2 – axis 2 encoder resolution.
Set to 0 if encoder is not used.
pulse_per_rev1 – axis 1 stepper pulse resolution.
This is used with encoder. If encoder is not used, set it to 0.
pulse_per_rev2 – axis 2stepper pulse resolution.
This is used with encoder. If encoder is not used, set it to 0.

Home Low Speed

int home_fl(int base, int axis, int dir);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1.
dir – 1 for positive and 0 for negative direction.

Home High Speed

int home_fh(int base, int axis, int dir);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1.
dir –1 for positive and 0 for negative direction

Home with Acceleration Profile

int home_ac(int base, int axis, int dir);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1.
dir – 1 for positive and 0 for negative direction.

Jog Low Speed

int jog_fl(int base, int axis, int dir);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1.
dir – 1 for positive and 0 for negative direction

Jog High Speed

int jog_fh(int base, int axis, int dir);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1.
dir – 1 for positive and 0 for negative direction

Jog with Acceleration Profile

int jog_accel(int base, int axis, int dir);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1.
dir – 1 for positive and 0 for negative direction

Immediate Stop

int immed_stop(int base,int axis);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1.

Decelerate Stop

int decel_stop(int base,int axis);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1.

11.2. Setup Command Functions

Set Low Speed, Low Speed, Acceleration, and S-Curve

int set_speed_accel_time(int base, int axis, unsigned long high_speed, unsigned long low_speed,unsigned long accel, int enable_scurve);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1.
high_speed – high speed. Range from 1 to 4M
low_speed – low speed. Range 1 to 4M
accel – acceleration time in milliseconds.
enable_scurve – 1 for enable S-curve and 0 for disable

Set current position

int setup_updown_counter(int base, int axis, long value);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1.
value – position value +/- 134M

Set Position Counter Input from Encoder or Pulse

int setup_pulse_counter_input_type(int base, int axis, int type);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1.
type – 0 for pulse input, 1 for encoder input

Set Encoder Multiplication Factor

int setup_encoder_mult(int base, int axis, int type);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1.
type – 0 for 1X, 1 for 2X, 2 for 4X encoder

Set Direction Output Signal Polarity

int setup_dir_logic(int base, int axis, int type);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1.
type – 0 or 1

Set Pulse Output Signal Polarity

int setup_pulse_logic(int base, int axis, int type);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1.
type – 0 or 1

Set Home Input Signal Polarity

int setup_home_sensor_logic(int base, int axis, int type);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1.
type – 0 for normally open, 1 for normally closed

Set Alarm Input Signal Polarity

int setup_alarm_sensor_logic(int base, int axis, int type);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1.
type – 0 for normally open, 1 for normally closed

11.3. Get Command Functions

Check Communication with the Controller Card
int check_com(int base);

Arguments:
base – base address of the controller card

Return:
0 – communication OK
-1 – no communication

Get Low Speed, High Speed, Acceleration, S-Curve
int get_speed_accel_time(int base, int axis, unsigned long *high_speed, unsigned long *low_speed,unsigned long *accel, int *enable_scurve);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1
*high_speed – high set speed
*low_speed – low set speed
*accel – acceleration time in milliseconds
*enable_scurve – 1 for S-curve, 0 for no Trapezoidal

Get Current Position Counter
long get_current_updown_counter(int base, int axis);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1

Return:
Current Position value

Get Current Pulse Rate
long get_current_pulse_rate(int base, int axis);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1

Return:
Current Pulse Rate value

Get Limit, Home, Alarm Input Status

int get_limits_home_state(int base,int axis);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1

Return:
Bit 0 - + Limit
Bit 1 - - Limit
Bit 4 – Home
Bit 6 - Alarm

Get Motor Status

int moving_state(int base,int axis);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1

Return:
1 – idle, not moving
3 – acceleration in progress
4 – Moving at High Speed
5 – Deceleration in Progress
6 – Moving at Low Speed

Get Position Counter Input either Encoder or Pulse

int get_pulse_counter_input_type(int base, int axis);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1

Return:
0 – pulse
1 – encoder

Get Direction Output Polarity
int get_dir_logic(int base, int axis);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1

Return:
0 or 1

Get Pulse Output Polarity
int get_pulse_logic(int base, int axis);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1

Return:
0 or 1

Get Alarm Input Polarity
int get_alarm_sensor_logic(int base, int axis);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1

Return:
0 or 1

Get Home Input Polarity
int get_home_sensor_logic(int base, int axis);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1

Return:
0 or 1

11.4. Driver Output Command Functions

Get Driver Enable Output State

int get_driver_enable_state(int base,int axis);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1

Return:
0 or 1

Set Driver Enable High

int driver_enable_high(int base,int axis);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1

Set Driver Enable Low

int driver_enable_low(int base,int axis);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1

Enable the output Driver Enable

int enable_driver(int base, int axis);

Arguments:
base – base address of the controller card
axis – axis number starting from 0 as axis 1

11.5. Digital IO Command Functions

Configure Digital IO as Inputs or Outputs

int config_dio(int base,int value);

Arguments:
base – base address of the controller card
value – 0x80 all outputs , 0x9B all inputs

Set Digital Output Port

int output_do_port(int base,int port, int value);

Arguments:
base – base address of the controller card
port – 0 to 2
value – 0x00 to 0xFF

Set Digital Output Bit

int output_do_bit(int base,int port, int bit, int state);

Arguments:
base – base address of the controller card
port – 0 to 2
bit – 0 to 7
value – 0x00 to 0xFF

Get Digital Input Port

int input_di_port(int base,int port);

Arguments:
base – base address of the controller card
port – 0 to 2

Get Digital Input Bit

int input_di_bit(int base,int port,int bit);

Arguments:
base – base address of the controller card
port – 0 to 2
bit – 0 to 7