MN100 Motion Controller Manual

Document Revision C4
May 16, 2018

MICROKINETICS CORPORATION
3380 Town Park Drive
Suite 330
Kennesaw, GA 30144
Tel: (770) 422-7845
Fax: (770) 422-7854
www.microkinetics.com

Table of Contents

1 Introduction

1.1 Features

The MN100 is a single axis stepper motor controller. It communicates with a master controller or a PC via RS485 serial port. The MN100 controller can be used with our DM8010, DR8010, DM4050, and UnoDrive or with any third party driver that accepts industry standard step and direction commands.

The features of the MN100 include:

- RS-485 interface to PC
- Up to 127 devices can be connected on a line
- Five software selectable baud rates (9600, 19200, 38400, 57600, and 115200)
- Convenient device addressing via on-board rotary dip switches
- Easy to use command set
- Connects directly to existing drivers
- Three user programs addressable by RS-485 command or button sequence

1.2 Specifications

Electrical Specifications

Maximum Step Rate	20,000 pulses/sec
Minimum Step Rate	20 pulses/sec
Operating Voltage	5 VDC ±5%
Current Requirements	100mA max.
Outputs	1 Step and 1 Direction (TTL compatible)
Inputs	6 TTL compatible (See Table 3)
I/O Ports	1 User Defined I/O (TTL compatible)
Output Current Drive	25 mA sink or source on each output
Physical Dimensions	2.235" w x 2.325" d x 0.75" h
Working Temperature Range	32°F ~ 158° F (0° C ~ 70° C)

2 Operation

The Motionet control system operation consists of sending commands via RS485 serial connection from the master which is typically a PC or panel mount LCD controller. The commands are received by the individual devices and acted upon. A response from each device is obtained upon receipt of a command and at the completion of each command with the exception of the special input which, if activated, will generate a message automatically. If a device receives an invalid command or it detects an error in transmission a code will be returned to the master indicating the error (see Section 2.2 Return Codes). Motion sequences can be stored in the on-board memory which provides space for two user programs and one 'Learn' program. The programs can be initiated by RS-485 command or by using the run switch and program sellect buttons.

2.1 User Programs

The MN100 stores all commands in a single file. Reserved program labels #PRGM1, #PRGM2, and #LEARN are used to denote the start and end of a "program" in the file. When the MN100 is instructed to execute it's stored program with the RS-485 'u' command or the local run switch, it will start at the first line in the file and stop execution when it reaches a reserved label or the end of the file. If the MN100 is instructed to execute a specific program, it will first locate the reserved label for that program and then start execution at the line following that reserved label. The execution will stop when it reaches the next reserved label or the end of the file.

NOTE: Be aware that entering learn mode will erase everything following the #LEARN label. For this reason, you should always store #PRGM1, #PRGM2, and any other important commands **before** the #LEARN label.

2.2 Program Execution (Local)

Programs can be executed locally using the run switch and Jog / Program Select buttons. The following button sequences are used to tell the MN100 which user program to execute:

Hold Down	Then Hold	Release	Release	To start execution at
Run Switch		Run Switch		Line 1
Run Switch	Jog -	Run Switch	Jog -	#PRGM1 Label
Run Switch	Jog +	Run Switch	Jog +	#PRGM2 Label
Run Switch	Jog - & +	Run Switch	Jog - & +	#LEARN Label

2.3 Learn Mode

A special button driven 'Learn Mode' allows the user to jog and store several positions in the on-board #LEARN program independant of a PC. The learn program can later be edited on a PC to add velocities, loops, and other advanced commands. The following proceedure describes learn mode operation:

1. Press & hold the **FEEDHOLD** button for 3 seconds
   (LED will flash to indicate that is has entered Learn Mode)
2. Use the **JOG+** and **JOG-** buttons to move to a desired position
   (LED will light solid while jogging)
3. Press & release **FEEDHOLD** to add the current position to the #LEARN program
   (LED will flash rapidly for 1 second to indicate it has stored the position)
4. Repeat steps 2 & 3 until all desired movements and positions have been stored
5. Press & hold the **FEEDHOLD** button for at least 5 seconds to exit Learn Mode
   (LED will double flash 3 times then stay solid to indicate it has left Learn Mode)

The positions are stored under the reserved label #LEARN in the MN100's memory. The first line will be a velocity command which will store the current jog speed, followed by motion commands with absolute positions.
NOTE: If the MN100 already contains motion commands after the #LEARN label, they will be overwritten as soon as you enter Lean Mode. It is recommended that you edit the program on a PC, after using learn mode, and move the commands above the #LEARN label to avoid accidental erasure.

2.4 Command Set

Table 1, on the following page lists the MN100 commands and their functions. Several commands are designated as **PROGRAM** or **REMOTE**. If the command is designated as program then that command is available only when the device is running a program from memory. The program commands will be accepted by the slave only when program mode is enabled. If the command is sent when in remote mode then an **"INVALID COMMAND"** message will be returned. If the command is designated as remote then that command will only function when operating in remote mode. If these commands are written to the device while in program mode then an **"INVALID PROGRAM COMMAND"** message will be returned and the command will not be stored in program memory.

Table 1 - Command Set

Command	Name	Parameter	Function
D	Delay	1 to 65535	Pauses for a number of milliseconds. 1000 = 1 sec
E	Read Position	None	Returns the position counter.
F	Profile Select	0 to 5	Selects the acceleration profile (See Table 4).
G	Jog Tap Steps	1 to 255	Set the number of steps per button push.
H	Hold	0 or 1	0 - disables synchronized moves between drivers. Execute moves immediately. 1 - enables synchronized moves between drivers. Waits for release command before executing move commands. (**Remote)**
I	Set Mode	0 or 1	0 - absolute mode (default) 1 - incremental mode
K	Release	None	Enables move commands if HOLD is enabled. (**Remote)**
L	Load Count	±8,388,607	Loads the position register with the specified data.
M	Move	±8,388,607	Performs an accelerated move generating the specified number of steps. Returns exit code if an error occurred or if the move was terminated by a switch closure.
N	Read Count	None	Returns the number of uncompleted steps.
O	Loop	Optional: 0 to 65535	Restart the program from beginning. (**Program**) Optionally specify number of loops, i.e. 'O3' loops 3 times, including the first run.
Q	Abort Move	None	Aborts the move in progress. (**Remote**)
R	Read Memory	None	Returns the program stored in memory.
S	Port Status	0 to 2	Returns status of the input ports (port 0 or port 2).
U	Run Program	Optional: 1 to 3	Runs the program stored in memory. (**Remote)** Optionally specify program to start: 'U1' – #PRGM1, 'U2' - #PRGM2, 'U3' - #LEARN
V	Velocity	20 to 20000	Sets the speed in steps/sec.
W	Wait	None	Halts program until input is low. (**Remote**)
Y	Jog Speed	20 to 20000	Sets the speed for jogging (not accelerated).
Z	Program Mode	None	Puts the device in program mode. All subsequent commands are written to memory until <Ctrl-D> is received. (**Remote**)
?	Firmware Revision	None	Returns the device name and firmware revision. (**Remote**)
*	Poll	None	Request response from all connected controllers. (**Remote**)
&	Retransmit	None	Request the MN slave to resend the last message. (**Remote**)
~	Change Baud Rate	0 to 4	Changes baud rate. Must be sent to all devices simultaneously (address 0) . 0 - 9600, 1 - 19200, 2 - 38.4K, 3 - 57.6K, 4 - 115K. (**Remote**)

!	Configure and Read/Write I/O Port	0 to 3	Controls reading and writing of I/O port. 0 - output at 0VDC (low) 1 - output at 5VDC (high) 2 - input to run program on switch closure (default) 3 - input with interrupt (high to low transition) (**Remote**)
<	Store Configuration	F,TTT,VVVVV,I	Stores startup settings for the MN100, where: F – Profile Select T – Jog Tap steps V – Jog Velocity I – I/O configuration (Must cycle power for settings to take effect)
#	Label	<text>	Marks a point in the program for branching. (**Program**) Special labels #PRGM1, #PRGM2, and #LEARN are used to specify the start of 3 separate programs. They must be all CAPS and can be addressed using the optional parameter of the 'U' command, or by pressing and holding 'Limit -', 'Limit +', or 'Limit - & +', before pressing the run switch.
^	Branch	<text> Optional: <text>,N (1 to 65535)	Transfers program execution to a line following a # with matching label. (**Program**) Optionally specify number of times to branch, i.e. '^MYBRANCH,3' will branch 3 times.

2.5 Return Codes

After a command is received the MN slave returns a single character code. Some of these codes acknowledge a command, others provide information, and others are error messages.

Table 2 lists the value, meaning, and description for each possible return code.

Table 2 - Return Codes

Value Returned	Meaning	Description
1	- Limit reached	Negative limit switch contacted during a negative direction move (motion interrupted)
2	+ Limit reached	Positive limit switch contacted during a positive direction move (motion interrupt)
3	Start	Command was received and is being processed.
4	Move Aborted	Move aborted by closure of the abort switch (motion interrupt).
5	Command Finished	The command has been processed.
6	Valid Data Returned	Data has been returned from MN slave.
7	Slave Checksum Error	The MN slave received a packet in which the received checksum does not match the calculated checksum (transmission error).
8	Slave Address Error	Master received a stop address from a MN slave that does not match the start address.
9	Label Not Found	A label was defined in the program but not found.
10	Program Abort	Program aborted by 'Q" command.
11	Running Program	Program running on MN slave.
12	Invalid Command	The MN slave received an invalid command.
14	Invalid Parameter	The parameter is invalid for the command.
15	No Command	No command in packet sent to MN slave.
16	No Move Pending	The MN slave received a release command but has not received a move command.
17	Move Already Pending	A move command was received by the MN slave but there is a move already waiting to be released. New command is ignored.
18	Baud Rate Changed	The baud rate changed correctly.
19	Slave Time-out	The MN slave did not receive a complete packet within the allotted time.
20	Busy	A move is in process and no commands can be received.
21	Master Buffer Overflow	The command string exceeded 11 bytes.
22	Present	The MN slave is present at specified address.
23	Master Receive Checksum Error	The Master received a packet in which the received checksum does not match the calculated checksum (transmission error).
24	Slave Buffer Overflow	The MN slave received more than 11 bytes.
25	Message Buffer Overflow	The hardware receive buffer on the MN slave received a new character before the previous one was retrieved.
26	Interrupt	A high to low transition was detected on the I.O pin (must be configured in this mode).
27	Master Time-out	Master did not receive a complete packet with in the allotted time.
28	Error	Master received invalid data from a MN slave.
29	Not Available	The specified port is unavailable.
30	Memory Full	Program memory is full and no more commands can be stored.
32	Move Stopped	Move stopped by 'Q' command (decelerated move).

2.6 Port Status

The bit assignments for each input port (connector H2) is shown in Table 3. If the corresponding bit is high then that input is active (switch is closed). The switches connected to these inputs should be normally open with one side connected to the input and the other side connected to ground (See application diagram figures 3 & 4). The pin connections for H2 are shown in Appendix A.

Table 3 - Bit Values

Port 0	
Bit	Function
0	Limit-
1	Limit+
2	On Fullstep
3	Fault
4	Abort
5	Feed-hold
6	N.A.
7	N.A.

Port 2	
Bit	Function
0	I/O
1	N.A.
2	N.A.
3	N.A.
4	N.A.
5	N.A.
6	N.A.
7	N.A.

Note: Port 1 is not available on the MN100.

2.7 Acceleration/Deceleration Profiles

The acceleration/deceleration profiles are trapezoidal with the acceleration and deceleration slopes being the same. There are four different profiles to select from and they are listed in Table 4. Typically one of the profiles is selected and all subsequent moves will accelerate/ decelerate according to the selected profile until a new profile is selected. The table number corresponds to the value to be entered with the profile select command (See Table 1 - Command Summary).

Table 4 - Profile Selections

Table	Start Speed (Steps/Sec.)	End Speed (Steps/Sec.)	Accel (Steps/Sec^2)
0	500	10000	2000
1	500	15000	5000
2	500	18000	8000
3	500	20000	10000
4	500	20000	20000
5	500	20000	40000

A graph of the acceleration/deceleration profiles are shown in Figure 1 below.

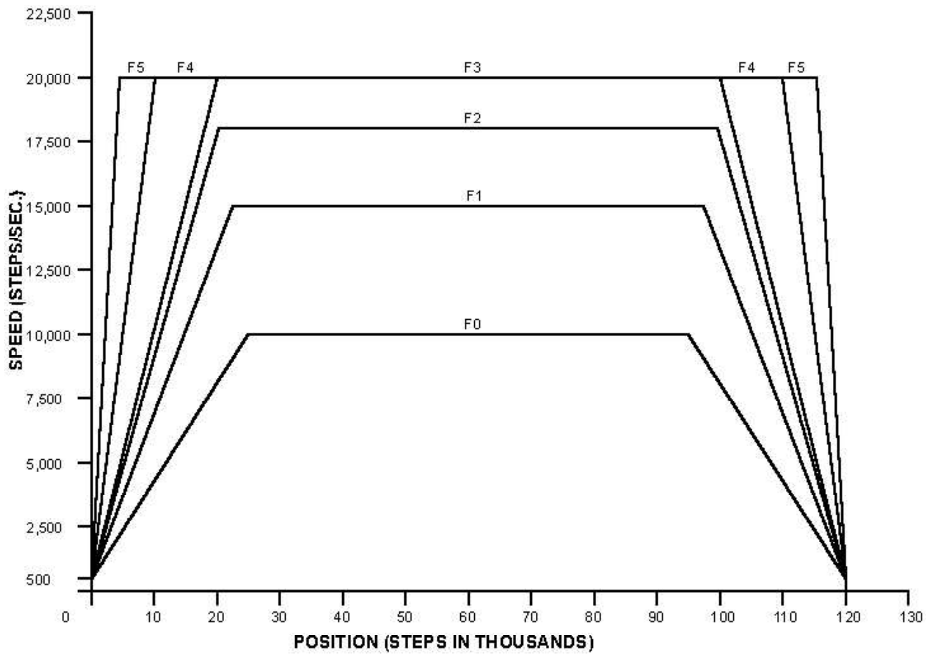


Figure 1 - Acceleration/Deceleration Profiles

2.8 Device Addressing

The address for each device is set via two rotary dip switches (SW1 and SW2). Each device must have a unique address between 1 and 127 decimal (1 - 7f hexadecimal). Setting the address for a device is accomplished by setting SW1 and SW2. Table 5 below shows the addresses and the corresponding switch settings.

Table 5 - Address Switch Settings.

Decimal	Hex	SW1	SW2
0	0	0	0
1	1	0	1
2	2	0	2
3	3	0	3
4	4	0	4
5	5	0	5
6	6	0	6
7	7	0	7
8	8	0	8
9	9	0	9
10	A	0	A
11	B	0	B
12	C	0	C
13	D	0	D
14	E	0	E
15	F	0	F
16	10	1	0
17	11	1	1
18	12	1	2
19	13	1	3
20	14	1	4
21	15	1	5
22	16	1	6
23	17	1	7
24	18	1	8
25	19	1	9
26	1A	1	A
27	1B	1	B
28	1C	1	C
29	1D	1	D
30	1E	1	E
31	1F	1	F
32	20	2	0
33	21	2	1
34	22	2	2
35	23	2	3
36	24	2	4
37	25	2	5
38	26	2	6
39	27	2	7
40	28	2	8
41	29	2	9
42	2A	2	A
43	2B	2	B
44	2C	2	C
45	2D	2	D
46	2E	2	E
47	2F	2	F
48	30	3	0
49	31	3	1
50	32	3	2
51	33	3	3
52	34	3	4
53	35	3	5
54	36	3	6
55	37	3	7
56	38	3	8
57	39	3	9
58	3A	3	A
59	3B	3	B
60	3C	3	C
61	3D	3	D
62	3E	3	E
63	3F	3	F
64	40	4	0
65	41	4	1
66	42	4	2
67	43	4	3
68	44	4	4
69	45	4	5
70	46	4	6
71	47	4	7
72	48	4	8
73	49	4	9
74	4A	4	A
75	4B	4	B
76	4C	4	C
77	4D	4	D
78	4E	4	E
79	4F	4	F
80	50	5	0
81	51	5	1
82	52	5	2
83	53	5	3
84	54	5	4
85	55	5	5
86	56	5	6
87	57	5	7
88	58	5	8
89	59	5	9
90	5A	5	A
91	5B	5	B
92	5C	5	C
93	5D	5	D
94	5E	5	E
95	5F	5	F
96	60	6	0
97	61	6	1
98	62	6	2
99	63	6	3
100	64	6	4
101	65	6	5
102	66	6	6
103	67	6	7
104	68	6	8
105	69	6	9
106	6A	6	A
107	6B	6	B
108	6C	6	C
109	6D	6	D
110	6E	6	E
111	6F	6	F
112	70	7	0
113	71	7	1
114	72	7	2
115	73	7	3
116	74	7	4
117	75	7	5
118	76	7	6
119	77	7	7
120	78	7	8
121	79	7	9
122	7A	7	A
123	7B	7	B
124	7C	7	C
125	7D	7	D
126	7E	7	E
127	7F	7	F

2.9 Data Packet Format

The following information is provided so you can write your own communications drivers. However, the MN Library includes routines that handle data transmission and reception without the need to write your own routines.

The data is transmitted in packets from the PC or master controller to each device. The packet consist of the address, a string of ASCII characters representing the command and data, and the checksum. The packet format for data being sent to a device is shown below.

Device address | Command | Parameter | Device address | Checksum

Note: Spaces are not allowed in the packet (command string).

Device address -	address of the device that is to receive the packet. The high bit of the address must be set to 1. The setting of the high bit of the address is handled by the MN Library routines.
Command -	a command that is listed in Table 1. Must be one of the commands listed in the table.
Parameter -	number required by the command (See Table 1 - Command Summary). The parameter must be in ASCII format with the value between 48 ('0') and 57 ('9'). Also, the data must be within the specified range for the command.
Checksum -	a 7-bit checksum that is the sum of all of the characters in the packet up to the checksum. This value is calculated by adding the values for each character in the packet then ANDing the checksum with 127 decimal (7f hexadecimal) to clear the high bit. Calculation of the checksum and clearing the high bit is handled by the MN Library routines.

example:
If sending the command "M1000" to device 5, the packet in ASCII representation (in hexadecimal) would appear as follows:

```
    M  1  0  0  0
85|4D|31|30|30|30|85|18
 |                 |  |---- Checksum
 |                 |-------- Device Address
 |-------------------------------- Device Address
```

The packet format for data being returned from a device is shown below.

Device address | Command | Return value | Return code | Device address | checksum

Device address - address of the device sending the packet to the PC or master controller.

The high bit of the address is set to 1 by the device.

Return value - Numerical value returned only if required by command.

Return code - single character value that indicates if an error has occurred or not (See Table 2 - Return Codes).

example:
If device number 5 is returning 5455 based on the Readcount ("E") command then the packet in ASCII representation (in hexadecimal) would appear as follows:

```
     E  5  4  5  5
85|45|35|34|35|35|06|85|28
 |                 |  |  |-Checksum
 |                 |  |-----Device Address
 |                 |---------Return Code
 |---------------------------------Device Address
```

3 Installation

3.1 MN100 Installation Diagram

The diagram below shows how to connect multiple MN100 units to the master controller. The RS485 is a multidrop configuration and each unit is connected to the same twisted pair cable as shown.

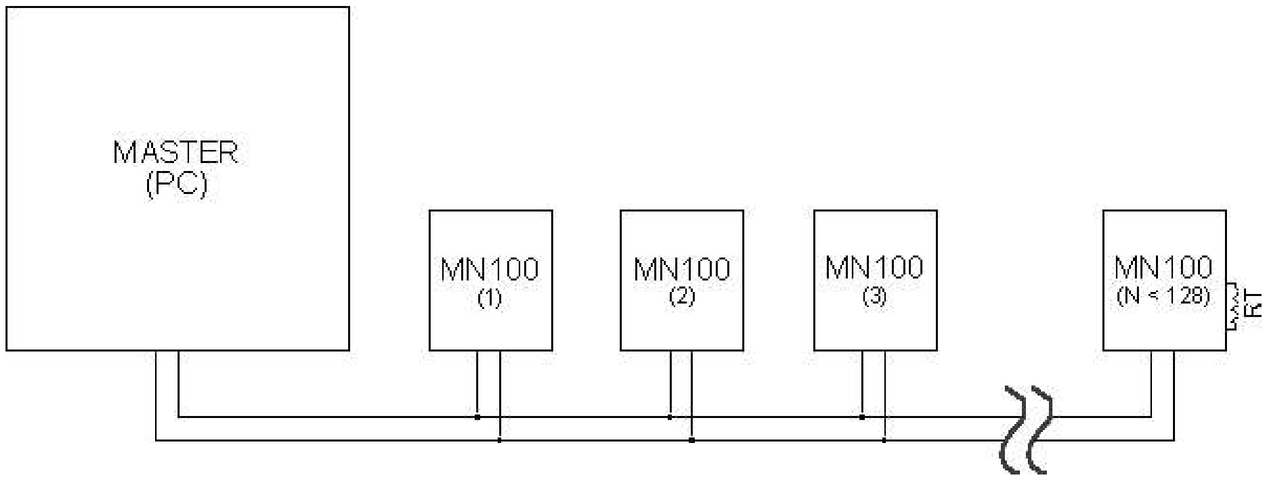


**Figure 2- Installation Diagram
Multiple MN100s**

NOTE: When installing multiple units make sure the termination resistor (RT) is removed from all of the devices except for the last one on the bus. If the resistors are not removed communications errors will occur.

3.2 MN100 Connection to UnoDrive

The MN100 can plug directly into an UnoDrive using connector H4. It can also, be wired to other drivers using screw terminal H1. When using an UnoDrive you must connect Vmm, Ground, and +5VDC to H3 (see Figure 1 below). When using any other driver you only need to connect +5VDC and Ground to H3 (See Figure 4 on the following page).

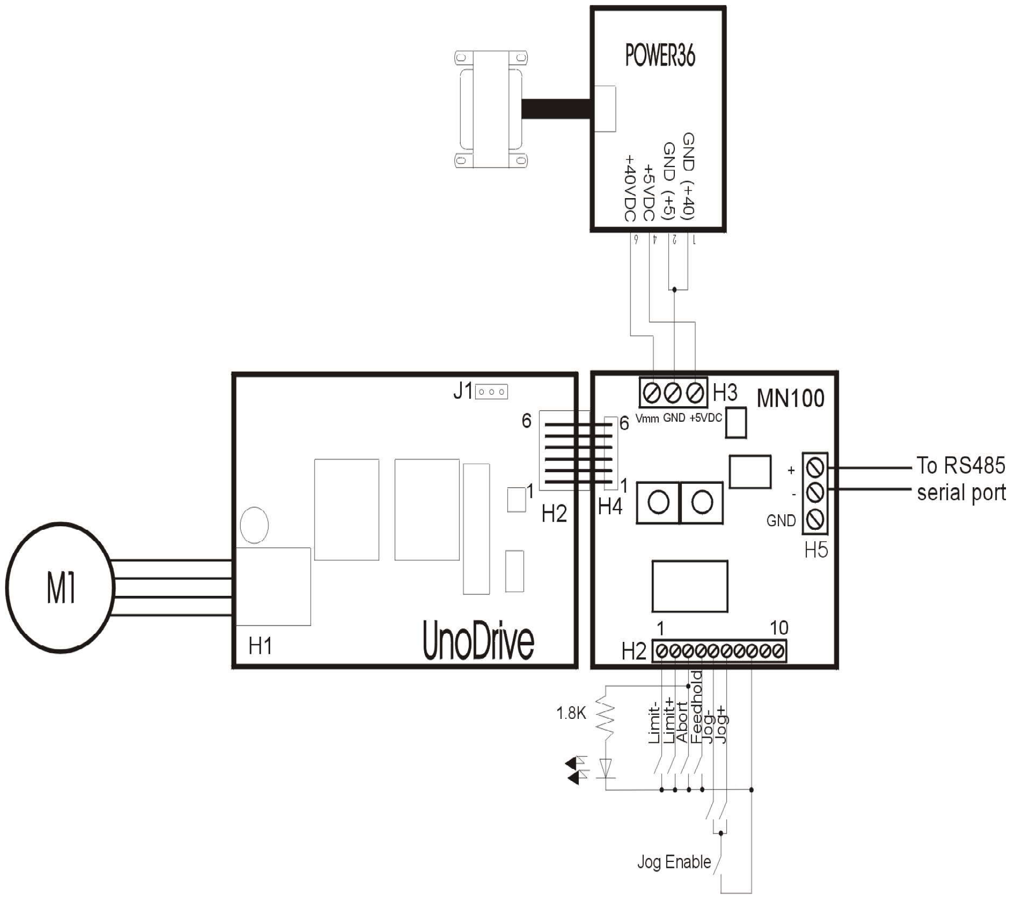


Figure 3 - Installation Diagram - UnoDrive

3.3 MN100 Connection to Other Drivers

The diagram below shows a typical wiring diagram for using the MN100 with most drivers. Note that H1 on the MN100 is connected one to one to the driver input connector on the DM4050, DR8010, and DM8010.

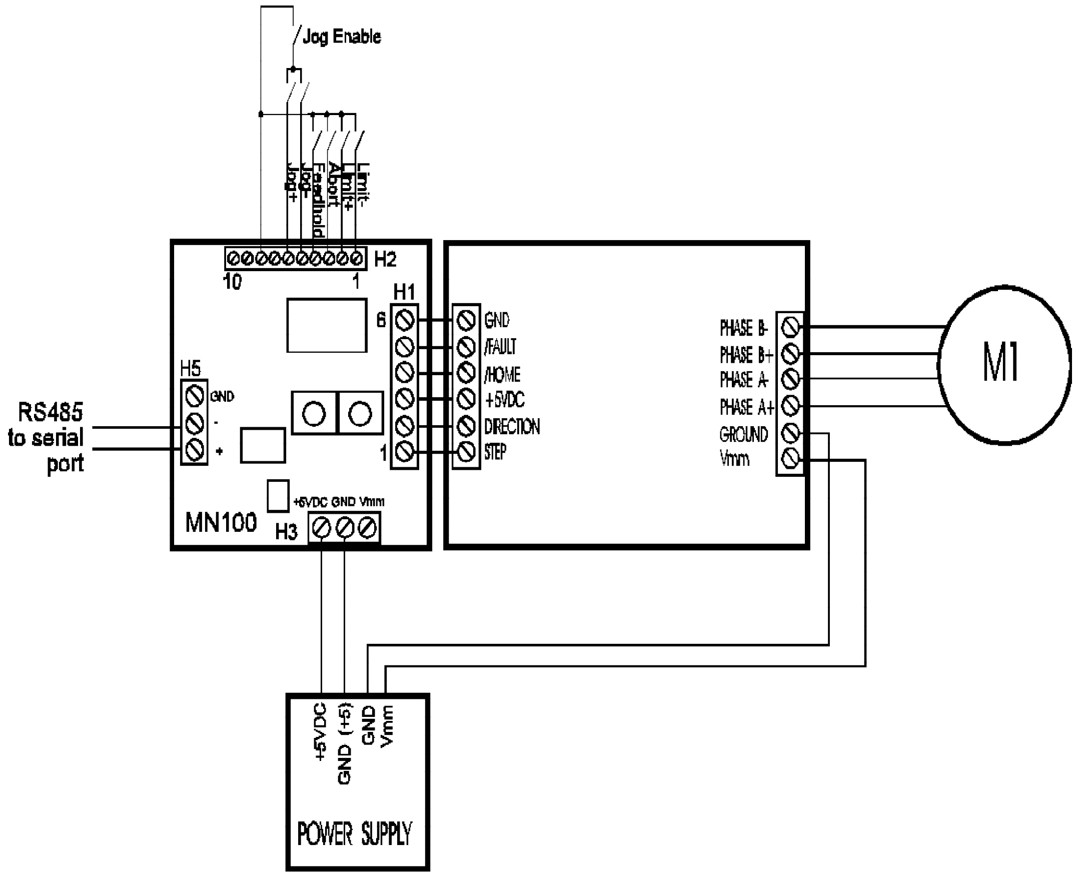


Figure 4 - Installation Diagram for DM4050, DM8010, and DR8010

4 Technical Support

Should you need help in identifying and correcting a problem, the MicroKinetics engineering staff is ready to assist you during business hours. You should refer to the documentation and verify any described adjustments before calling. Be prepared to supply the model number of all components and any software and/or dip switch or jumper settings.

4.1 How to Obtain Technical Support

Technical support is available as follows:

Via Email
Email MicroKinetics with a description of problem symptoms to `helpdesk@microkinetics.com` where it is reviewed and answered daily.

Via Fax
Fax a detailed description of the problem to 770-422-7854 including your fax and voice number. An engineer will call to help you.

Via Telephone
Call our main line directly and request Hardware Tech Support. The number is 770-422-7845.

4.2 Product Return Procedure

The technical support staff can determine if the problem requires returning the product for testing and can give you an RMA (Return Merchandise Authorization) number to write on the outside of the package for proper routing. This improves repair turnaround time.

When returning an electronic product, always pack in the original antistatic bag. If original packaging is not available, wrap in aluminum foil and place in container to withstand shipping and handling. Always insure product with shipping company for full value.

If a product is returned to us for repair, is tested and found to operate within the rated specifications, a nominal testing fee will apply. Please inquire as to the testing charge at the time you obtain the RMA number.

5 Software

5.1 Windows 95 DLL

This section describes the routines in the 32-bit DLL (MNW.DLL). This software is supplied with the MN100 and runs under Win95. These functions can be called from the DLL using LoadLibrary and GetProcAddress as shown below.

```
// Load and get the address of the DLL.
hLibrary = LoadLibrary("MNW.DLL");
if (hLibrary == NULL)
{
        MessageBox(NULL, "Could not open DLL - MNW.DLL", NULL, MB_OK);
        return false;
}
// Get the address of the specific procedure you wish to call.
mnInitCommDLL = (INITCOMMPROC)GetProcAddress(hLibrary, "mnInitComm");
if (!mnInitCommDLL)
{
        MessageBox(NULL, "Cannot load mnInitComm from DLL", NULL, MB_OK);
        return false;
}
// Call the function
rtn_code = (mnInitCommDLL)(&port, &rate, &size);
// Close the library
FreeLibrary(hLibrary);
```

The function names are defined as extern "C" to avoid name mangling and the __declspec(dllexport) WINAPI declaration is used. Also, the names are case sensitive and do not have a leading underscore.

To interface to the DLL use the type definitions and declarations shown below. The type definitions are defined in the MNTERMW.H header file supplied with the MN100 software. For an example on how to use these functions in Visual C from Borland see the C++ programs shown on the distribution disk for the MNTERMW.EXE program.

```
typedef int (*CHKRXDQUEPROC)(int *);
typedef int (*CLOSECOMMPROC)(int *);
typedef int (*INITCOMMPROC)(int *, int *, int *);
typedef int (*POLLDEVICESPROC)(int *, int *, int *, char *, int *);
typedef int (*PROCESSCHARPROC)(char *, int *, int *, char *);
typedef int (*SETBAUDPROC)(int *, int *);
typedef int (*VERSIONPROC)(int *);
typedef int (*XMITPACKETPROC)(char *, char *, int *);
```

```
HINSTANCE hLibrary;
CHKRXDQUEPROC        mnChkRxDQueDLL;
CLOSECOMMPROC        mnCloseCommDLL;
INITCOMMPROC         mnInitCommDLL;
POLLDEVICESPROC      mnPollDevicesDLL;
PROCESSCHARPROC      mnProcessCharDLL;
SETBAUDPROC          mnSetBaudDLL;
VERSIONPROC          mnVersionDLL;
XMITPACKETPROC       mnXmitPacketDLL;
```

You can also convert the DLL to a library and use the library at link time (see the documentation for the compiler you are using). If you use the library, you still need to distribute the MNW.DLL along with the EXE. The library only provides linkage information for the functions in the DLL and it does not include the actual code from the DLL.

```
extern "C" int __declspec(dllimport) WINAPI mnChkRxDQue(int *);
extern "C" int __declspec(dllimport) WINAPI mnCloseComm(int *);
extern "C" int __declspec(dllimport) WINAPI mnInitComm(int *, int *, int *);
extern "C" int __declspec(dllimport) WINAPI mnPollDevices(int *, int *, int *, char *, int *);
extern "C" int __declspec(dllimport) WINAPI mnProcessChar(char *, int *, int *, char *);
extern "C" int __declspec(dllimport) WINAPI mnSetBaud(int *, int *);
extern "C" void __declspec(dllimport) WINAPI mnVersion(int *);
extern "C" int __declspec(dllimport) WINAPI mnXmitPacket(char *, char *, int *);
```

The following section describes each of the routines This section describes the routines in the Windows DLL supplied with the MN100. Each function in the DLL returns a value >= 0 on success, and < 0 if a communications error occurred. The only exception is mnVersion which does not have a return value. Also, see the MNTERMW.H header files for the definitions for the com ports, baudrates, and errors. The descriptions follow the format outlined below.

Purpose	Describes the use of the procedure.
Syntax	Shows the proper syntax for calling the procedure using C/C++.
Prototype	Function prototype for the DLL and Library.
Parameters	Describes each parameter used in the calling syntax.
Example	Shows the use of the routine in a typical code fragment.

mnChkRxDQue

Purpose mnChkRxDQue determines if data has been received. Returns 1 if data has been received, returns 0 otherwise.

Syntax
```
DLL: data_avail = (ChkRxDQueDLL)(&port);
Library: data_avail = mnChkRxDQue(&port);
```

Prototype
```
DLL: typedef int      (*CHKRXDQUEPROC)(int *);
Library: extern "C" int __declspec(dllimport) WINAPI mnChkRxDQue(int *);
```

Parameters port - value of selected com port.

Example
```
int data_avail;
int port = COM1;

data_avail = (ChkRxDQueDLL)(&port);
if (data_avail)
{
        ......
}
OR
if ((ChkRxDQueDLL)(port))
{
        .......
}
```

mnCloseCommPort

Purpose To close the open com port. This must be done to release the com port and interrupt when exiting your program.

Syntax
```
DLL: rtn_code = CloseComPortDLL(&port)
Library: rtn_code = mnCloseComPort(&port);
```

Prototype
```
DLL: typedef int      (*CLOSECOMMPROC)(int *);
Library: extern "C" int __declspec(dllimport) WINAPI mnCloseComm(int *);
```

Parameters port - value of selected com port.

Example
```
int port = COM1;

rtn_code = (CloseComPortDLL)(&port);
```

mnInitCommPort

Purpose Initializes the selected com port.

Syntax DLL: rtn_code = (InitCommPortDLL)(&port, &baudrate, &buffer);
Library: rtn_code = mnInitComPort(&port, &baudrate, &buffer);

Prototype DLL: typedef int (*INITCOMMPROC)(int *, int *, int *);
Library: extern "C" int __declspec(dllimport) WINAPI mnInitComm(int *, int *, int *);

Parameters port - value of selected com.
baudrate - value at which to set the com port baudrate.
buffer - size of receive and transmit buffers

Example

```
int port  = COM1;
int baudrate = B9600;       // set for 9600 baud
int buffer = Size128;
int rtn_code;

rtn_code = (InitComPortDLL)(&port, &baudrate, &buffer);
if (rtn_value > 0)
{
        printf("Could not initialize specified com port.");
        exit(0);
}
```

mnPollDevices

Purpose Polls all devices on the bus and sets a corresponding flag in the device_flag array for each active device that is found.

Syntax DLL: rtn_code = (PollDevicesDLL)(device_flag, &port, &device_count, message, &buffer);
Library: rtn_code = mnPollDevices(device_flag, &port, &device_count, message, &buffer);

Prototype DLL: typedef int (*POLLDEVICESPROC)(int *, int *, int *, char *, int *);
Library: extern "C" int __declspec(dllimport) WINAPI mnPollDevices(int *, int *, int *, char *, int *);

Parameters device_flag - integer array, = 1 if device is present at that address 0 otherwise.
port - value of selected com.
buffer - size of receive and transmit buffers
message - error message returned by function.
device_count - number of devices found

If return value is < 0 then a communication error has occurred.

Example

```
int device_flag[128];
int port = COM1;
int buffer = Size128;
int device_count;
char message[80];

rtn_code = (PollDevicesDLL)(device_flag, &port, &device_count, message,
&buffer);
if (device_flag[1] == 1)
        printf("Device found at address 1");
else
        printf("No device found at address 1");
```

mnProcessChar

Purpose To process the incoming characters and respond accordingly.

Syntax DLL: rtn_code = (ProcessCharDLL)(input_string, &port, rec_address, message);
Library: return_code = mnProcessChar(input_string, &port, rec_address, message);

Prototype DLL: typedef int (*PROCESSCHARPROC)(char *, int *, int *, char *);
Library: extern "C" int __declspec(dllimport) WINAPI mnProcessChar(char *, int *, int *, char *);

Parameters input_string - string returned from the device.
port - value of selected com port.
rec_address - address of the device sending the information.
message - error message returned by function.

return_code - indicates the response of the device . The function returns 0 if a complete string has not been received. Otherwise it returns a code indicating the status/error (See Table 2 - Return Codes). The codes are defined MN_x.H.

Example

```
int port = COM1;
char input_string[15];
char rec_address[2];
char message[80];

return_code = (ProcessCharDLL)(input_string, &port, rec_address, message);
switch(return_code)
{
        ....
}
```

mnSetBaud

Purpose Sets the baudrate for the open COM port

Syntax DLL: rtn_code = (SetBaudDLL)(&port, &baud);
Library: rtn_code = mnSetBaud(&port, &baud);

Prototype DLL: typedef int (*SETBAUDPROC)(int *, int *);
Library: extern "C" int __declspec(dllimport) WINAPI mnSetBaud(int *, int *);

Parameters port - value of selected com port
baud - integer indicating desired baudrate

Example

```
int port = COM1;
int baud = Baud9600;

rtn_value = (SetBaudDLL)(&port, &baud);
```

mnVersion

Purpose Gets the version number of the DLL being used.

Syntax DLL: (VersionDLL)(&revision);
Library: mnVersion(&revision);

Prototype DLL: typedef int (*VERSIONPROC)(int *);
Library: extern "C" void __declspec(dllimport) WINAPI mnVersion(int *);

Parameters revision - is an integer that is 100 times the revision number. i.e. 310 for rev. 3.1.

Example

```
int revision;

(VersionDLL)(&reversion);
printf("Driver Version: %4.1f\n", (double)revision/100);
```

mnXmitPacket

Purpose	Transmits the command packet to the addressed device.

Syntax	DLL: rtn_code = (XmitPacketDLL)(command_string, &address, &port);
	Library: rtn_code = mnXmitPacket(command_string, &address, &port);

Prototype	DLL: typedef int (*XMITPACKETPROC)(char *, char *, int *);
	Library: extern "C" int __declspec(dllimport) WINAPI mnXmitPacket(char *, char *, int *);

Parameters	command_string - command string to be transmitted. address - address of device to send packet to. port - value of selected com port

Example	int port = COM1;
	char command_string[15];
	char address;
	
	rtn_code = (XmitPacketDLL)(command_string, address, port);

5.2 Windows Terminal Program - MNTERMW.EXE

The Win95 terminal program (MNTERMW.EXE) provides an interface to the MN100 devices. There are several parts to the terminal program including Command Editor windows, Program Editor windows, and a Device selection window. The Command Editor windows have a Device, Port, Baud, Window, and Help menu when it is active. The Program Editor adds two menus when it is active - Edit and Program. The user can open multiple command editors and program editors and have each window address a different device.

When the MNTERMW program is started it initializes the com port, polls for active devices and opens a command window with the address of the first device found.

NOTE: The MNTERMW program matches its baudrate with the baudrate of the connected devices at start-up. Also, all of the connected devices must be operating at the same baudrate. The power-up baudrate of the MN100 is 9600.

Device Menu

Save	Saves the current configuration for the port and baud rate.
PollDevices	Sends out the poll command (*) to determine which devices are active
Select Device	Allows the user to select a different device address for the active window.
Exit	Exits the program when selected.

Port Menu

The port menu allows the user to select the COM port to use. This value is the first parameter stored in the MNTERMW.DEF configuration file. This file can be changed using a text editor. The values for the ports is shown below.

COM1 - 0
COM2 - 1
COM3 - 2
COM4 - 3

Baud Menu

The baud menu allows the user to select the baud rate to use. This value is the second parameter stored in the MNTERMW.DEF configuration file. The values for the ports is shown below.

9600	5
19200	6
38400	7
57600	8
115200	9

Edit Menu

Cut	Cuts the selected text from the window
Copy	Copies the selected text to the clipboard.
Paste	Paste the copied text at the current cursor position.
Select All	Highlights all of the text in the active window.
Delete	Deletes the selected text.

Program Menu

These items apply to the active window and the device it addresses.

Read	Reads the program from the device.
Write	Writes the window buffer to the MN100 program memory.
Clear	Clears the window buffer.
Execute	Executes the program stored in program memory.
Home	Returns the motor to the home position.
Stop	Stops program execution (decelerates to stop).

Window Menu

Tile	Tiles the open windows.
Cascade	Cascades the open windows.
Arrange Icons	Arranges icons of minimized windows.
New Command Editor	Opens a new command editor window.
New Program Editor	Opens a new program editor window.
Close	Closes the active window.

Help Menu

About	Shows about box for program.

5.3 DOS Library Routines

This section describes the routines in the DOS Libraries supplied with the MN100 . The descriptions follow the format outlined below.

Purpose	Describes the use of the procedure.
Syntax	Shows the proper syntax for calling the procedure using C/C++.
Prototype	Shows the prototype for the function (Supplied in mnterm.h).
Parameters	Describes each parameter used in the calling syntax.
Example	Shows the use of the routine in a typical code fragment.

mnChkRxDQue

Purpose mnChkRxDQue determines if data has been received. Returns 1 if data has been received, returns 0 otherwise.

Syntax data_avail = mnChkRxDQue(port);

Prototype int mnChkRxDQue(int);

Parameters port - value for selected com port.

Example

```
int data_avail;
int port = COM1;

data_avail = mnChkRxDQue(port);
if (data_avail)
{
    ......
}

OR

if (mnChkRxDQue(port))
{
    .......
}
```

mnCloseComPort

Purpose	To close the open com port. This must be done to release the com port and interrupt when exiting your program.
Syntax	mnCloseComPort(port);
Prototype	int mnCloseComPort(int);
Parameters	port - value of selected com.

Example

```
int port = COM1;

mnCloseComPort(port);
```

mnInitComPort

Purpose	mnInitComPort initializes the selected com port.
Syntax	rtn_value = mnInitComPort(port, irq, baudrate, buffer);
Prototype	mnInitComPort(int, int, int, int)
Parameters	port - value of selected com. irq - interrupt number to use with the selected com port. baudrate - value at which to set the com port baudrate. buffer - size of receive and transmit buffers

Example

```
int port  = COM1;
int irq = 4;
int baudrate = B9600;   // set for 9600 baud
int buffer = Size128
int rtn_value;

rtn_value = mnInitComPort(port, irq, baudrate, buffer);
if (rtn_value > 0)
{
    printf("Could not initialize specified com port.");
    exit(0);
}
```

mnPollDevices

Purpose	mnPollDevices polls all devices on the bus and sets a corresponding flag in the device_flag array for each active device that is found.
Syntax	mnPollDevices(device_flag, port, buffer, message)
Prototype	int mnPollDevices(int *, int, int, char *);
Parameters	device_flag - integer array containing the flag value for each device. If the flag for a given device is 1 then a device is present at that address. port - value of selected com. buffer - size of receive and transmit buffers message - error message returned by function.

Example

```
int device_flag[128];
int port = COM1;
int buffer = Size128;
char message[80];

mnPollDevices(device_flag, port, buffer, message);
if (device_flag[1] == 1)
    printf("Device found at address 1");
else
    printf("No device found at address 1");
```

mnProcessChar

Purpose To process the incoming characters and respond accordingly.

Syntax return_code = mnProcessChar(input_string, port, rec_address, message);

Prototype int mnProcessChar(char *, int, int *, char *);

Parameters input_string - string returned from the device.
port - value of selected com.
rec_address - address of the device sending the information.
message - error message returned by function.

return_code - indicates the response of the device . The function returns 0 if a complete string has not been received. Otherwise it returns a code indicating the status/error (See Table 2 - Return Codes). The return codes are defined in the MN_x header files. If data is received, the "input_string" holds the command followed by the ASCII data.

Example

```
int port = COM1;
char input_string[15];
char rec_address[2];
char message[80];

return_code = mnProcessChar(input_string, port, rec_address, message);
switch(return_code)
{
    ....
}
```

mnVersion

Purpose mnVersion returns the version number of the software that is being used.

Syntax mnVersion(&revision);

Prototype mnVersion(int *)

Parameters revision - is an integer that is 100 times the revision number. i.e. 310 for rev. 3.1. To use the number as a string, use the following formula.

Example

```
int drvrev;

mnVersion(&drvrev);
printf("Driver Version: %4.1f\n", (double)drvrev/100);
```

mnXmitPacket

Purpose	Transmit the command packet to the addressed device.
Syntax	mnXmitPacket(command_string, address, port, buffer);
Prototype	mnXmitPacket(char *, char, int, int);
Parameters	command_string - command string to be transmitted. address - address of device to send packet to. port - value of selected com port buffer - size of receive and transmit buffers

Example

```
int port = COM1;
int buffer = Size128;
char command_string[15];
char address;

mnXmitPacket(command_string, address, port, buffer);
```

5.4 DOS Terminal Program - MNTERM.EXE

Study and use this code for reference. Feel free to copy and paste into your program any portion you need.

The terminal program controls communication between the PC and the Motionet series of motion controllers. It takes the input typed in by the user and assembles it into a packet along with the board address and checksum and sends the packet out the serial port. The device that has the matching address responds to the command and returns a START code to indicate it received the data. The device then processes the data and either returns the required data or a code (See Table 2 - Return Codes for a description of each code) indicating it completed the command or that there was an error.

Once a device is busy it will not accept any other command except the abort move 'Q' command. If any other command is received during a move, the addressed device will return a code indicating that it is busy. If a command is sent to a device while it is processing the previous command (other than the move command) the new command is ignored. Make sure you wait for the return code or data following the START code before sending the same device another command.

The baud rate can be changed from the terminal program. The baud rates supported are 9600, 19200, 38400, 57600 and 115200. The default baud rate is 9600. To change the baud rate type <#> followed by the number corresponding to the desired baudrate (See selections below).

0	9600 baud
1	19200 baud
2	38400 baud
3	57600 baud
4	115200 baud

To communicate with a different device, type <@> followed by the device number (0 - 127). If a command is sent to address 0 all of the connected devices will respond to it. None of the devices should have their addresses set to 0.

5.5 Distribution Disk

The files listed in each section below are included on the distribution disk. Create a directory on your hard drive and copy the files from the appropriate directory on the floppy disk to the directory you created.

5.5.1 Win95

ABOUT1.CPP	1,000	09-19-97	5:40p
ABOUT1.H	1,029	09-19-97	5:17p
COMENTRY.CPP	9,754	10-07-97	12:19p
COMENTRY.H	3,173	10-06-97	3:09p
DEV_SEL.CPP	3,005	10-07-97	11:30a
DEV_SEL.H	1,018	09-25-97	12:03p
MNFORM.CPP	31,231	10-08-97	5:21p
MNFORM.H	4,362	10-06-97	3:09p
MNTERMW.CPP	3,206	10-08-97	5:51p
MNTERMW.H	5,747	10-08-97	5:21p
MNTERMW.EXE	328,704	10-08-97	5:29p
MNTERMW.MAK	2,046	10-08-97	3:22p
MNTERM .DEF	6	10-08-97	3:38p
MNW.DLL	254,464	10-08-97	5:32p
PRG_EDIT.CPP	10,757	10-07-97	12:22p
PRG_EDIT.H	3,923	10-07-97	11:48a
WSC.DLL	119,808	09-15-97	3:26p

NOTE: You must distribute MNW.DLL and WSC.DLL with your EXE.

5.5.2 DOS

MNTERM.H	6,585	10-08-97	4:41p
MNTERM.C	20,509	10-08-97	4:42p
MNTERM.EXE	55,707	10-08-97	4:49p
MN_S.H	9,733	10-08-97	4:37p
MN_S.LIB	23,040	10-08-97	4:44p
MN_M.H	9,797	10-08-97	4:37p
MN_M.LIB	23,552	10-08-97	4:44p
MN_C.H	9,720	10-08-97	4:36p
MN_C.LIB	23,552	10-08-97	4:44p
MN_L.H	9,809	10-08-97	4:36p
MN_L.LIB	23,552	10-08-97	4:44p
FEEDLEN.C	6,849	07-10-97	10:47a
FEEDLEN.EXE	36,313	07-10-97	10:24a
SHAPEFRM.C	6,847	07-10-97	10:44a
SHAPEFRM.EXE	36,759	07-10-97	10:49a
SWTESTER.C	6,461	07-10-97	10:53a
SWTESTER.EXE	27,448	07-10-97	10:56a

6 Applications

6.1 Feed to Length

The MN100 can be used in a feed to length application by specifying the following parameters.

Distance
Number of times to repeat operation

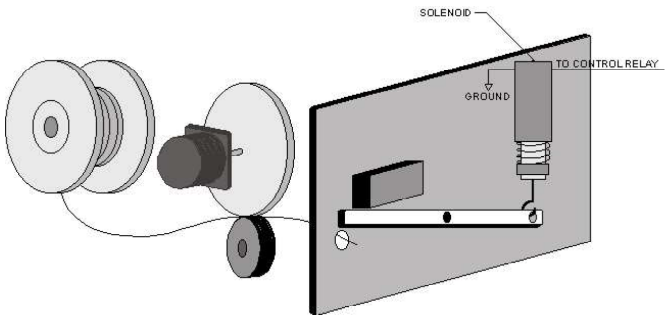


Figure 5 - Feed to Length

The following is a source listing of a feed to length application. You need to change the following definitions to match your application and recompile the program.

DRIVER_RES
REVS_PER_INCH
STEP_RATE

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
//      This program performs a Feed to length to length operation.
//      The inputs are distance, driver resolution, revolutions/inch and
//      the number of times to perform the operation.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#include    <stdio.h>
#include    <conio.h>
#include    <dos.h>
#include    <stdlib.h>
#include    <string.h>
#include    "mn_cs.h"

#define     MOTOR_RES           200
#define     DRIVER_RES          8           // driver resolution (steps/fullstep)
#define     REVS_PER_INCH       10.0f       // number of turns of motor to move device an inch
#define     STEP_RATE           5000        // steps/sec
// see mn_cs.h for definitions
#define     PORT_ID             PORT0       // COM1 - change for your system
#define     BAUDRATE            B9600       // 9600 Baud (default for MN100)
#define     INT_IRQ             IRQ4        // use IRQ4
```

```
COM   com_id;
float distance;
int   i, device_flag[MAX_BOARDS];
char  command_string[15], data_string[15];

int   process_rtn_packet(COM, long *);
void  perform_operation(long, int, COM);

void main()
{
      long  data, steps;
      int   device_address, num_operations;

      textmode(C80);
      clrscr();

      com_id = init_com_port(PORT_ID, INT_IRQ, BAUDRATE);
      if (com_id == NULL)
      {
            printf("\n\r    Could not initialize com port.");
            printf("\n\r    Press any key to exit program");
            getch();
            clrscr();
            exit(0);
      }
      printf("\n\r    Checking for an active device.");
      check_devices(device_flag, com_id);
      device_address = 0;
      // If multiple devices are connected and active, the
      // device with the lowest address will be used.
      for (i = 0; i < MAX_BOARDS; i++)
      {
            if (device_flag[i])
            {
                  device_address = i;
                  break;
            }
      }
      if (!device_address)
      {
            printf("\n\r    No active device found.  Press any key to exit");
            close_com_port(com_id);
            getch();
            clrscr();
            exit(0);
      }
      clrscr();
      send_packet("f1", device_address, com_id);    // set profile
      process_rtn_packet(com_id, &data);

      sprintf(command_string, "v%d", STEP_RATE); // set velocity
      send_packet(command_string, device_address, com_id);
      process_rtn_packet(com_id, &data);

      send_packet("w1", device_address, com_id);// I/O is output @ logic 1 (off)
      process_rtn_packet(com_id, &data);

      printf("Enter distance to move (inches): ");
      scanf("%f", &distance);
      printf("Enter number of operations: ");
      scanf("%d", &num_operations);
```

```
        clrscr();
        steps = (long)(distance * REVS_PER_INCH * DRIVER_RES * MOTOR_RES);

        for (i = 0; i < num_operations; i++)
        {
                printf("\n\r%d: ", i+1);
                perform_operation(steps, device_address, com_id);
        }

        close_com_port(com_id);
}
void perform_operation(long steps, int address, COM id)
{
        long    data;

        send_packet("l0", address, id);  // load counter with 0
        process_rtn_packet(id, &data);

        sprintf(command_string, "m%ld", steps);// perform move
        send_packet(command_string, address, id);
        process_rtn_packet(id, &data);

        send_packet("e", address, id);   // get absolute counter value
        process_rtn_packet(id, &data);
        if (data != steps)              // compare steps moved to steps commanded
        {
                printf("\n\rMove not completed."); // if != move not completed
                printf("\n\rPress any key to continue.");
                getch();
        }
        send_packet("w0", address, id);   // Turn on output
        process_rtn_packet(com_id, &data);
        printf("Output ON! ");

        delay(1000);            // delay 1 second

        send_packet("w1", address, id);   // Turn off output
        process_rtn_packet(com_id, &data);
        printf("Output OFF!");
}
int process_rtn_packet(COM id, long *data)
{
        int     return_code, start_flag;
        int     i, rec_address[2];
        char    input_string[15];

        while(1)
        {
                if (chk_RxD(id))        // check receive buffer for data
                {
                        return_code = process_char(input_string, id, rec_address);
                        if (return_code == 0)
                                continue;
                        else if (return_code == START)
                                continue;
                        else if (return_code == DONE)
                                break;
                        else if (return_code == VALID_DATA)
                        {
                                i = 1;
                                while (input_string[i] != 0)
                                {
                                        data_string[i-1] = input_string[i];
                                        i++;
                                }
                                *data = atol(data_string);
                                break;
                        }
                        else if (return_code < 0x20)
                        {
                                printf("\n\rError detected or move stopped");
                                printf("\n\rby external switch closure.");
                                break;
                        }
                }
        }
        return(return_code);
}
```

6.2 Switch Tester

The MN100 can be used to test switches. Two examples for implementing the testing are shown below.

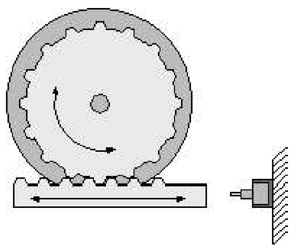


Figure 6 - Rack and Pinion

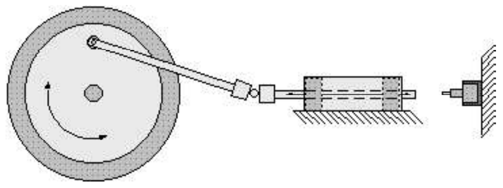


Figure 7 - Universal Joint

The following is a source listing of a switch tester application using the setup shown in Figure 7 above. You may need to change the following definitions to match your application.

DRIVER_RES
STEP_RATE

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//          SWTESTER.C - Switch Tester
//          Date:04-16-97
//
//          This program can be used with a MN100 to test pushbutton switches.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#include     <stdio.h>
#include    <conio.h>
#include    <dos.h>
#include    <stdlib.h>
#include    <string.h>
#include    "mn_cs.h"

#define     MOTOR_RES           200
#define     DRIVER_RES          2           // driver resolution (steps/fullstep)
#define     STEP_RATE           400         // steps/sec

// see mn_cs.h for definitions
#define     PORT_ID             PORT0       // COM1 - change for your system
#define     BAUDRATE            B9600       // 9600 Baud (default for MN100)
#define     INT_IRQ             IRQ4        // use IRQ4

COM  com_id;

float   distance;

int     i, device_flag[MAX_BOARDS];

unsigned char       input_stat, last_status;

char    command_string[15], data_string[15];
```

```
int     process_rtn_packet(COM, long *);
void    perform_operation(long, int, COM);

void main()
{
        float   dist_left;
        long    data, steps;
        int     device_address, num_operations;
        int     ret, key;

        textmode(C80);
        clrscr();

        com_id = init_com_port(PORT_ID, INT_IRQ, BAUDRATE);

        if (com_id == NULL)
        {
                printf("\n\r     Could not initialize com port.");
                printf("\n\r     Press any key to exit program");
                getch();
                clrscr();
                exit(0);
        }
        printf("\n\r     Checking for an active device.");
        check_devices(device_flag, com_id);

        device_address = 0;

        // If multiple devices are connected and active, the
        // device with the lowest address will be used.
        for (i = 0; i < MAX_BOARDS; i++)
        {
                if (device_flag[i])
                {
                        device_address = i;
                        break;
                }
        }
        if (!device_address)
        {
                printf("\n\r     No active device found.  Press any key to exit");
                close_com_port(com_id);
                getch();
                clrscr();
                exit(0);
        }
        clrscr();
        send_packet("f1", device_address, com_id);     // set profile
        process_rtn_packet(com_id, &data);

        sprintf(command_string, "v%d", STEP_RATE); // set velocity
        send_packet(command_string, device_address, com_id);
        process_rtn_packet(com_id, &data);

        send_packet("w2", device_address, com_id);   // set I/O as input
        process_rtn_packet(com_id, &data);

        printf("Position plunger using the jog buttons so that the button");
        printf("on the switch is fully depressed.  Press any any when done.");
        getch();
```

```
	send_packet("l0", device_address, com_id);	// init counter to 0
	process_rtn_packet(com_id, &data);

	steps = DRIVER_RES * MOTOR_RES / 2; // steps per 0.5 revolutions

	send_packet("s2", device_address, com_id); // read input status
	ret = process_rtn_packet(com_id, &data);

	last_status = (unsigned char)data & BIT0;

	sprintf(command_string, "m%ld", steps);// rotate motor 1/2 rev.

	while(1)
	{
		if (kbhit())
		{
			key = getch();
			if (key == ESC)
				break;
		}
		else
		{
			clrscr();
			printf("\n\r	Moving");
			send_packet(command_string, device_address, com_id);
			ret = process_rtn_packet(com_id, &data);
			clrscr();
			printf("\n\r	Reading Switch");
			delay(250);
			send_packet("s2", device_address, com_id); // read input status
			ret = process_rtn_packet(com_id, &data);

			if (ret == VALID_DATA)
			{
				input_stat = (unsigned char)data & BIT0;
				if (input_stat == last_status)
				{
					printf("\n\r	switch defective");
					break;
				}
				else
					last_status = input_stat;
			}
			else if (ret == DONE)
				continue;
			else if (ret < 0x20)
				printf("\n\r	Error in transmission - %x.", ret);
			send_packet("l0", device_address, com_id);	// zero position counter
			process_rtn_packet(com_id, &data);
		}
	}
	close_com_port(com_id);
}
int process_rtn_packet(COM id, long *data)
{
	int	return_code, start_flag;
	int	i, rec_address[2];
	char	input_string[15];

	while(1)
	{
```

```
                if  (chk_RxD(id))       // check receive buffer for data
                {
                        return_code = process_char(input_string, id, rec_address);
                        if (return_code == 0)
                                continue;
                        else if (return_code == START)
                                continue;
                        else if (return_code == DONE)
                                break;
                        else if (return_code == VALID_DATA)
                        {
                                i = 0;
                                while (input_string[i+1] != 0)
                                {
                                        data_string[i] = input_string[i+1];
                                        i++;
                                }
                                data_string[i] = 0;
                                *data = atol(data_string);
                                break;
                        }
                        else if (return_code < 0x20)
                                break;
                }
        }
        return(return_code);
}
```

6.3 Shape Forming System

Multiple MN100s can be used to bend various materials into a specific shape. An example of this type of application is shown below.

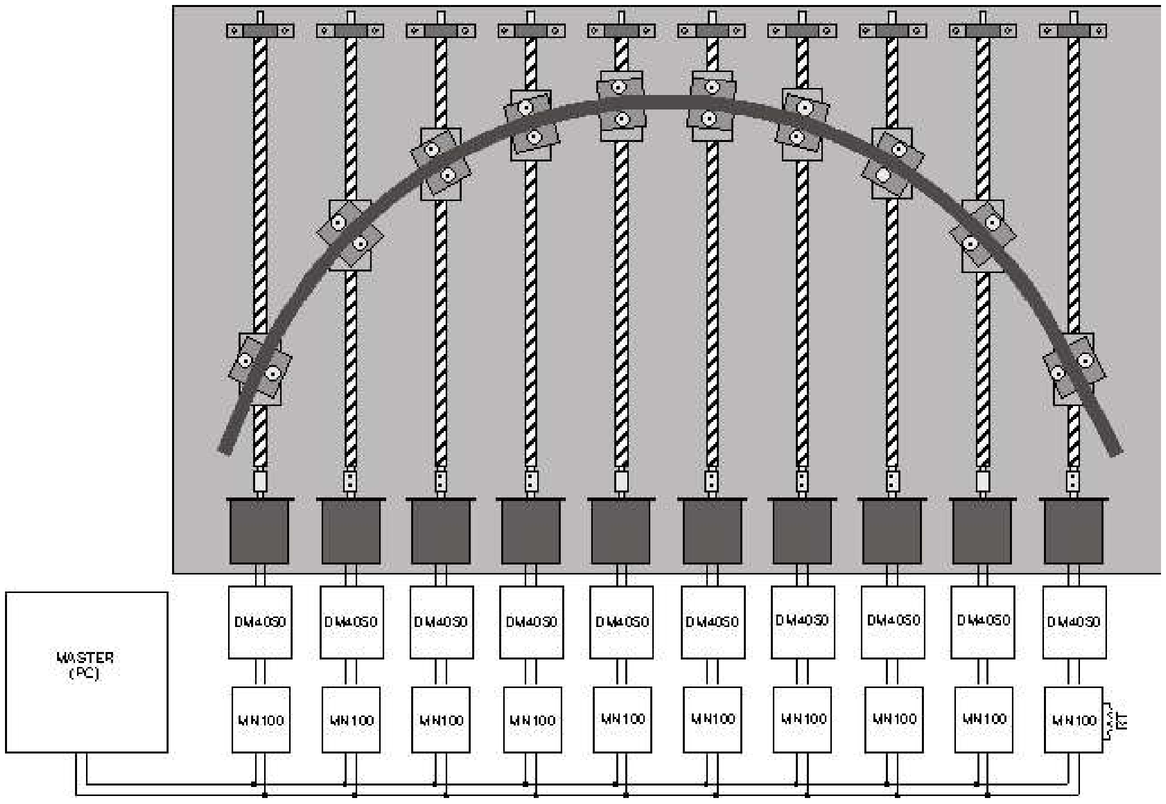


Figure 8 - Shape Forming

The following is a source listing of a shape forming application using the setup shown above. You may need to change the following definitions to match your application.

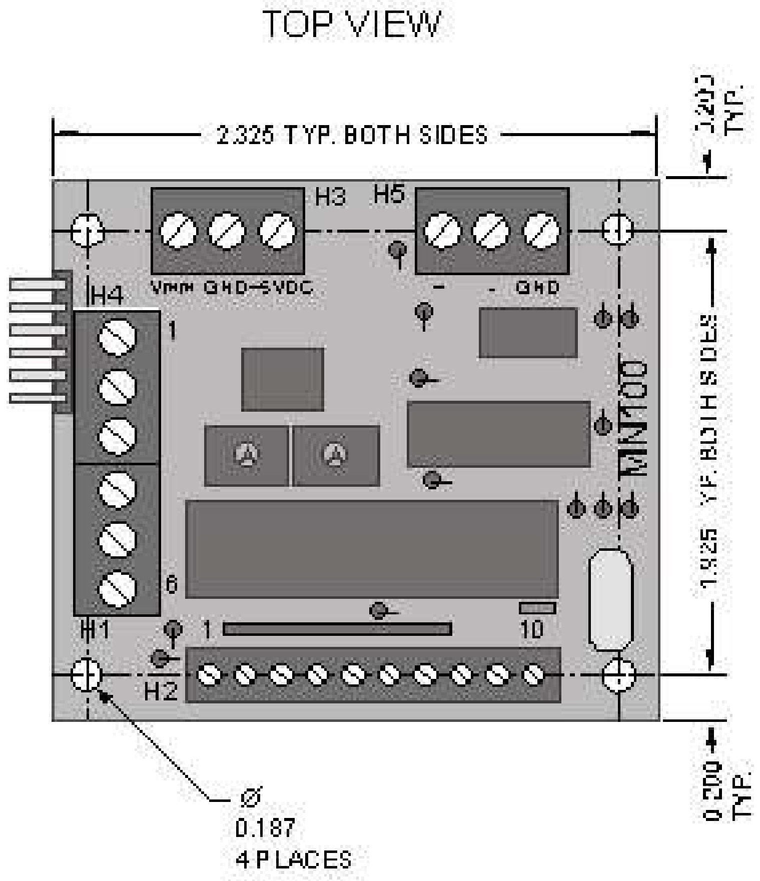
DRIVER_RES
REVS_PER_INCH
STEP_RATE

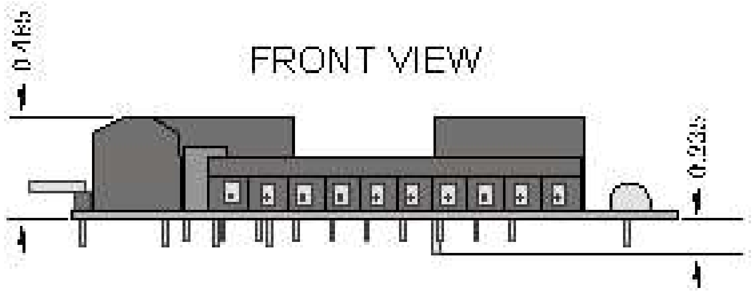
```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//              SHAPEFRM.C - Shape Former example
//              Date:04-16-97
//
//              This program can be used with multiple MN100s to bend materials
//              into a specific shape.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#include        <stdio.h>
#include        <conio.h>
```

```
#include      <dos.h>
#include      <stdlib.h>
#include      <string.h>
#include      <ctype.h>
#include      <math.h>
#include      "mn_cs.h"

#define       MOTOR_RES              200
#define       DRIVER_RES             8             // driver resolution (steps/fullstep)
#define       REVS_PER_INCH          10.0f         // number of turns of motor to move device an inch
#define       STEP_RATE              4000          // steps/sec
// see mn_cs.h for definitions
#define       PORT_ID                PORT0         // COM1 - change for your system
#define       BAUDRATE               B9600         // 9600 Baud (default for MN100)
#define       INT_IRQ                IRQ4          // use IRQ4
#define       PI                                   3.141592654f

COM           com_id;
double        angle_inc;
float         distance;
int           i, device_flag[MAX_BOARDS];
char          command_string[15], data_string[15];

int     process_rtn_packet(COM, long *);
void    perform_operation(long, int, COM);

void main()
{
        long    data, steps[MAX_BOARDS];
        int     device_address[MAX_BOARDS], num_operations;
        int     device_count;
        int     key;

        textmode(C80);
        clrscr();

        com_id = init_com_port(PORT_ID, INT_IRQ, BAUDRATE);

        if (com_id == NULL)
        {
                printf("\n\r     Could not initialize com port.");
                printf("\n\r     Press any key to exit.");
                getch();
                clrscr();
                exit(0);
        }
        clrscr();
        printf("\n\r     Checking for an active device.");
        check_devices(device_flag, com_id);
        device_count = 0;
        for (i = 0; i < MAX_BOARDS; i++)
                if (device_flag[i])
                        device_address[device_count++] = i;
        if (!device_count)
        {
                printf("\n\r     No active device found.  Press any key to exit");
                close_com_port(com_id);
                getch();
                clrscr();
                exit(0);
        }
        send_packet("f1", 0, com_id);       // set profile
        process_rtn_packet(com_id, &data);

        sprintf(command_string, "v%d", STEP_RATE); // set velocity
        send_packet(command_string, 0, com_id);
        process_rtn_packet(com_id, &data);

        send_packet("l0", 0, com_id);  // initialize position counter to 0
        process_rtn_packet(com_id, &data);

        send_packet("i0", 0, com_id);       // set for absolute mode
        process_rtn_packet(com_id, &data);

        send_packet("h1", 0, com_id);  // enable release mode
        process_rtn_packet(com_id, &data);

        send_packet("w1", 0, com_id);  // set I/O as output and turn off
        process_rtn_packet(com_id, &data);

        clrscr();
        printf("\n\r    Enter maximum distance to move (inches): ");
        scanf("%f", &distance);

        angle_inc = 180.0f/(device_count+1);

        for (i = 0; i < device_count; i++)
        {
                steps[i] = (long)(distance * REVS_PER_INCH * DRIVER_RES * MOTOR_RES);
                steps[i] = (long)(steps[i] * sin(PI/180.0f*angle_inc*(i+1)) + 0.5f);
        }
        for (i = 0; i < device_count; i++)
        {
                sprintf(command_string, "m%ld", steps[i]);
                send_packet(command_string, device_address[i], com_id);
                process_rtn_packet(com_id, &data);
        }
        clrscr();
        printf("\n\r     Positioning motors.");
        send_packet("k", 0, com_id);  // enable release mode
        process_rtn_packet(com_id, &data);
        send_packet("w0", 0, com_id);  // turn on output
        process_rtn_packet(com_id, &data);
        printf("\n\r     Output ON!");
        delay(5000);
        send_packet("w1", 0, com_id);  // turn off output
        process_rtn_packet(com_id, &data);
        printf("\n\r     Output OFF!");
        printf("\n\r     Repositioning to start.");
        send_packet("m0", 0, com_id);  // return to start
        process_rtn_packet(com_id, &data);
        send_packet("k", 0, com_id);  // send release command
        process_rtn_packet(com_id, &data);
        send_packet("h0", 0, com_id);  // enable release mode
        process_rtn_packet(com_id, &data);

        close_com_port(com_id);
}
int process_rtn_packet(COM id, long *data)
{
        int      return_code, start_flag;
        int      i, rec_address[2];

        char     input_string[15];
        while(1)
        {
                if (chk_RxD(id))        // check receive buffer for data
                {
                        return_code = process_char(input_string, id, rec_address);
                        if (return_code == 0)
                                continue;
                        else if (return_code == START)
                                continue;

                        else if (return_code == DONE)
                                break;

                        else if (return_code == VALID_DATA)
                        {
                                i = 1;
                                while (input_string[i] != 0)
                                {
                                        data_string[i-1] = input_string[i];
                                        i++;
                                }
                                *data = atol(data_string);
                                break;
                        }
                        else if (return_code < 0x20)
                        {
                                printf("\n\rError detected or move stopped");
                                printf("\n\rby external switch closure (%d).", return_code);
         getch();
                                break;
                        }
                }
        }
        return(return_code);
}
```

Appendix

Apendix A - Mechanical Drawing





Apendix B - Connector Pin Descriptions

Pin #	**H1 Description (MN100X only)**
1	Step
2	Direction
3	+5VDC
4	ON Fullstep
5	Fault
6	Ground

Pin #	**H2 Description**
1	-Limit
2	+Limit
3	Abort
4	Feedhold
5	-Jog
6	+Jog
7	I/O or Run
8	Ground
9	Ground
10	+5VDC

Pin #	**H3 Description**
1	Vmm
2	Ground
3	+5VDC

Pin #	**H4 Description (MN100U only)**
1	Vmm
2	Ground
3	Ground
4	+5VDC
5	Step
6	Direction

Pin #	**H5 Description**
1	RS485+
2	RS485-
3	Ground