



MechaView

for MB3001

Version 1.0

Revision	ECO No.	Description	Date	Appd.
-		Preliminary	03/01/06	LM

Table of Contents

Introduction

Terminal versus Bus Mode
Normal Initialization

Motor Operation

Direction naming convention
Position Resolution
Step calculation
Step Accuracy
Motor Power Management
Hardware Limit Detection

Fast Start

Getting Started

Command Summary

Command Syntax

Control Commands

Unit Select
Emergency Stop

Indexing Commands

Index Flag Levels
Index Home Position
Index Motor Right
Index Motor Left
Index Maximum Position

Motor Move Commands

Move Absolute
Move Relative

Power Commands

Set Hold Power
Set Run Power

Read Commands

Reads ADC Port
Read I/O Port

Read Status

Set Motor Operation Parameters

Set Fast (Slew) Speed

Set Step Mode

Set Ramp Rate

Set Slow Speed

Set Unit Parameters Command

Set Bus Mode

Set Unit ID

Restore Unit State

Save Unit State

Write Data Commands

Write Pins

Modifying the Program

The monitor and Dispatch routines

Adding custom commands

Programming Examples

Initialization

Indexing the motor

Running a program

Board setup

Minimum Configuration

Trouble Shooting

Common Problems, and solutions

Updating the Program

Where to find updates

Using AVRISP

Disclaimer

License

Conditions of Use

GNU GENERAL PUBLIC LICENSE

Motor Operation

Motion - Direction Naming Convention

The software is designed with the assumption that a stepping motor will typically be used to drive right hand lead screw drive. Clockwise (CW) movement of the motor (facing the shaft like a clock face) increments the position count (moves the nut away from the motor), and is referred to RIGHT motion. Counter Clockwise (CCW) movement decrements the position count (moves the nut towards the motor), and is referred to as LEFT motion.

Position Resolution

The internal positioning unit is a step. The bit resolution of a step is 32 bits of full step position and 8 bits of sub-step position. Data is formatted as:

STEP (4bytes) + SUB-STEP (1byte)

Absolute Position is always a positive number, from 0 (the absolute CCW limit) to 4,294,967,295.0 (FFFFFFFF.00 hex) the absolute CW limit. The default origin at reset is half way in between or 2147483648+0 (80000000+0 hex). The origin can be set to an arbitrary position with the "IH" command.

The sub-step value is a binary fraction, and is either 0.0 or 0.5 (0x80 hex). 0.5 is one half step. In Full Step Mode, the sub-step value is always 0. All fractions entered are automatically converted to 0.5, which is the only resolution available in the MB3001 controller.

Maintaining a sub-step fractional value allows the MB3001 to switch stepping modes while maintaining position accuracy. In addition position values are compatible with MechaBlox units such as the MB3003 which is capable of 8 levels of micro-stepping.

Step Calculation

Two calculated motion commands are available, Move Absolute and Move Relative. Internally, the calculated step count is a relative displacement to the current position.

In the case of the Move Relative command, if the count is positive, the motor moves CW, If the count is negative, the motor moves CCW. In the case of MA, the sign, if entered is ignored.

The position unit is expressed in full steps +fraction. The motor mode determines if the count is in half step, or in full step. Depending upon the motor mode, the actual step

count, the number of motor “pulses” required to move to the position will be different.

For compatibility, the Move Absolute (MA) command is always expressed in full steps -even in half step mode. That is, there is no difference when entering `MANnnn` in either full step or half step mode. The only difference is in half step mode, you can enter a fractional step as either a `.0`(optional) or a `.5` i.e. `MANnnn.0` or `MANnnn.5`. Entering a fractional step in full step mode will cause no error, but it will be ignored.

Note: When the position is reported with the Return Status (RS) command, the position step and fraction is expressed as `NNNNNNNN+F`. This is because the period '.' symbol is the ACK response. An embedded period would complicate automatic control software design.

In the case of the Move Relative (MR) Command, relative move steps are always expressed as a count of the current step type. For example, a `MRnnnn` command in half step mode is equivalent to a `MR(nnnn/2)` command in full step mode, but at half speed since twice as many steps are taken to move the same distance. Also, because internally all calculation are done with 32bit numbers, when in half step mode the effective range of motion is 0 to 2,147,483,648 in terms of full steps, since there are twice as many steps taken to move the same distance. Entering a relative move outside this range will result in an exception.

Switching between full step mode and half step mode will cause no positioning errors, but note that the in-between half step positions are of lower power than full step positions because only one motor phase is active. Also note that when switching between half stepping and full stepping, if the current step position is a half step, the first “full step” will automatically be a half step to re-sync the motor phase to the next full power step.

For compatibility, IM and IH commands only take full step values. Entering a fractional step will cause no error, but it will be ignored.

Step Accuracy

In general, for a unipolar motor step errors are non-cumulative and cancel after 4 steps. However, step accuracy will vary depending upon the motor current, acceleration, running speed and deceleration control values. These are highly motor and system dependent.

Full and Half stepping modes are compatible, and can be switched with loosing step accuracy. Wave step is not compatible with Full or Half Step Mode. Switching from Wave step to Full or Half step will result in step position error as the motor re-syncs its step phases.

Motor Power Management

Motor power is automatically switched between different running and hold current settings by the MB3001 controller. The running current and hold current are programmable and controlled by a 6-bit digital to analog converter (DAC). The levels are set by the **PR** and **PH** commands.

Default Hardware Limit Detection

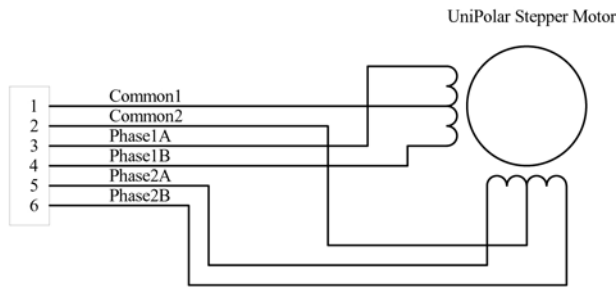
Hardware limit detection is always enabled. At reset, the left and right default limit detect level is a logic low. This is compatible with a normally open (n.o.) mechanical limit switch, or no switch connected at all. If you are using a break-beam photo-interrupter, which is normally on, you will need to program the limit detect level to a logic high before attempting to move the motor. See the **IF** - Index Flag Level command.

Fast Start

Getting the MB3001 Running

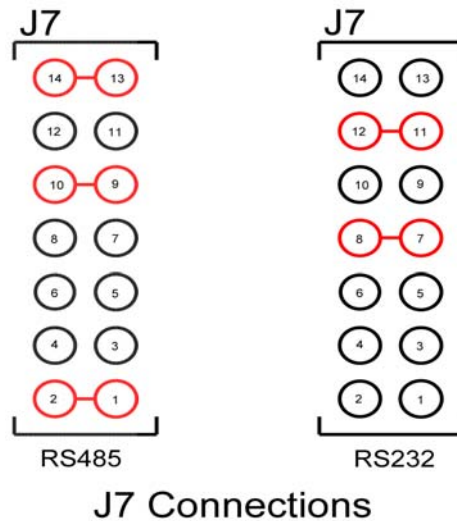
The MB3001 comes factory programmed and set up for terminal operation. All that is required is an 8-30VDC power supply, an RS232C terminal, or PC with a suitable communication program, and a Stepper Motor with a mating connector (Molex/Waldom 22-01-3067) installed as shown in figure 3.2.7 from the MB3001 the hardware manual.

Figure 3.2.7 Stepper Motor Conenction



The MB3001 comes with a DC power cable, and a RS232C adapter cable. Connect the DC power cable into either CN8 or CN9, and the RS232C cable into CN1. Set the Baud rate at the terminal for 19,200 Baud, 8 bit data, no parity bit, and 1 stop bit.

If the MB3001 board has been previously used in RS485 mode, be sure that jumpers 7-8, 11-12 are reinstalled on J7 for terminal operation.



MB3001 V1.0 MechaView Command Summary

Command	Cmd	Ext	Dat1	Dat2	Rtn	
Control Commands						
Unit select	@		8	-	A/N	Select addressed unit
Emergency Stop	X					Halts the motor and turns off power
Indexing Commands						
Index Flag Levels	I	F	8	-	A/N	Set Limit and Index signal active levels
Index Motor Home Position	I	H	32	-	A/N	Set home position count
Index Motor CCW (left)	I	L	-	-	A/N	Move motor to left limit, zero position
Index Max Travel	I	M	32	-	A/N	Set Max Travel (Right Limit Count)
Index Motor CW (Right)	I	R	-	-	A/N	Move motor to right limit, set position
Motor Move Commands						
Move Absolute	M	A	32	-	A/N	Move to absolute count position
Move Relative	M	R	32	-	A/N	Move to position +/- count
Power Commands						
Set Hold Power	P	H	8	-	A/N	Set idle current level
Set Run Power	P	R	8	-	A/N	Set run current level
Read Commands						
Read ADC	R	A	-	-	16	Reads ADC Channel, returns 10bit value
Read Driver	R	D			8	Reads drive pin levels
Read Register	R	I	8	-	8	Reads a register or I/O port
Read Status	R	S	-	-	SB	Returns status block
Read Version	R	V	-	-	V	Returns version and release data
Motor Settings Commands						
Set Fast (slew) Speed	S	F	16	-	A/N	Set fast pulse rate
Set Step Mode	S	M	16	-	A/N	Set full, half or wave mode
Set Ramp Rate	S	R	16	-	A/N	Set ramp-up intervals
Set Slow Speed	S	S	16	-	A/N	Set slow pulse rate
Set Unit Parameter Commands						
Set Bus Mode	U	B	8	-	A/N	Set Full or Half Duplex mode
Set Unit ID	U	I	8	-	A/N	Set unit ID
Restore Unit State	U	R	-	-	A/N	Restore all settings
Save Unit State	U	S	-	-	A/N	Save status and operating settings
Write Data						
Write Driver	W	D	8	-	A/N	Set drive pin level

Command Syntax

The basic command syntax is:

command extension parameter c/r

where command and extension are letters and parameter is a number followed by a carriage return (c/r = hex value 0x0d) - with no spaces between the data. With the exception of the halt command X, the commands and extensions are not case sensitive.

The parameter format is [{qualifier} number]

[number] = required parameter
{qualifier} = optional qualifier

For terminal compatibility and ease of programming, the number is an ASCII encoded BCD number (Binary Coded Decimal) and is one of 3 types:

byte = 8bit value (a value from 0 to 255)
integer = 16bit value (a value from 0 to 65535)
long = 32bit value (a value from 0 to 4,294,967,295)

The qualifier is an optional negative sign "-"

In the event that a larger number than specified by the command parameter is entered, for example, an integer instead of a byte, no error is generated, but only the lowest part of the number will be used. In this case the lower byte of the integer will be used as input data.

When entering data, a carriage return will terminate the data string. Leading zeros do not have to be entered for 32bit data (10 BCD digits). Do not use comma separators. If all 10 digits are entered, a C/R is not required for entry.

At any point of command or data entry, if the space-bar (hex 0x20) is pressed the input is discarded, and the unit will return a NAK (if in terminal mode)

Control Commands

Address Unit

Usage: @[ID]
ID is a byte value from 1 to 31 (0x01..0x1f).
Returns :ACK/NAK

Description:

This commands selects the addressed unit, it is active only in bus mode. In a bus system, each unit must have a unique ID. When a unit is addressed, it will process all following commands until a @ commands selects another unit.

Bus operation differs from terminal operation in two ways. First, data is not automatically acknowledged, and second commands may be sent in a block. The current selected unit will receive and buffer the commands, then execute then in the sequence they are received.

Example:

To address a unit of ID, enter:

@

Enter new [BYTE] (0..255)

BYTE c/r

The unit will NOT echo back

Emergency Stop

Usage: X (must be uppercase)

Description:

The MB3001 maintains an input buffer. Commands are normally processed in the order in which they are received. The X command is execute a system emergency stop upon receipt.

The motor if running, is stopped, the power level is set to off, or lowest setting depending upon the motor type, and the controller unit performs a "hot reset."

After an emergency stop, if the motor was moving – especially at slew speed nothing should

McGourty Associates, LLC

be assumed. It is important to go through a complete re-initialization procedure.

For an orderly exit under normal conditions, see the **US** command.

Example:

To emergency stop enter:

X c/r

Units will NOT echo back

Indexing commands

Introduction

There are two "index" counts maintained by the controller, motor position limits by Left limit (always 0) and the Right limit step count, and the steps per revolution count of the motor shaft. The process of setting these counts is referred to as indexing.

Motor Position Limit Indexing.

Internally, the MB3001 controller maintains the location of the motor position as a displacement from a left stop position value always assumed to be 0, and a programmable right stop position. The controller will not allow the motor step past these limit values. At reset the right position stop value defaults to 4,294,967,295.

Some applications may require external restraint of motor motion. Left and Right Limit sensor flags can be used detect to maximum travel position in the left (CCW) and right (CW) directions. The default left position index count is 0, and the right index count is the current step count at the point that the right limit flag was detected.

Indexing the motor is done by stepping the motor to the right or left at slow speed until the corresponding index flag is set. Indexing is automatically done using full step mode, the most accurate step mode.

If the motor is indexed left, the current position count is set to 0. When the motor is indexed right, the current count is first cleared before stepping begins. When the right index flag is detected, the cumulative step value is save as the right limit.

To set the limits, first use the Index Left command to get the zero position, then the Index Right command to maximum right direction step count.

The flags are normally used to set stepping limits, but also serve as a safety stop in the even that the motor drops steps for what ever reason.

If you are not using limit detection, be careful of an accumulated step count in one direction exceeding the limit count in either direction.

Set Index Flag Signal Levels

Usage: IF[level]
where level is a byte
Returns ACK/NACK

An index or limit signal is simply a signal generated at specific angular rotation points, or at fixed linear displacements. There are three signals detected by the MB3001 controller, Left limit (Index Left), Right Limit (Index Left) and shaft position (Shaft Index).

The controller uses the left and right index flags two ways. First, at initialization, the flags can be used to set absolute step values left and right. The controller will then use these values when calculating moves so that it does not exceed the absolute allowed step displacements. For example, if a command is given to exceed an absolute limit, the controller calculates the move to the limit, moves there, and stops without step errors. However, an exception will be returned and an set in the status word.

The controller continuously monitors the index flag status when moving. If an index flag is unexpectedly detected when moving in the flags direction, stepping will be halted immediately in that direction, and will not be allowed until the flag is cleared. Stepping in the opposing direction (as long as the opposing index flag is clear) is not affected. This allows the controller to back away from the the tripped index flag, and clear the error.

Depending upon the sensor type, the active level which defines the point of detection may be a logic "High" or a logic "Low" signal. The IF command is used to set the active signal level. The lower three bits of the upper status word byte is the index and limit active level settings.

```
;* hi_byte ; f e d c b a 9 8
; 1 1 1 1 1 1 1 1
; | | | | | | | | CW Limit (right) (0|1)
; | | | | | | | | CCW Limit (left) (0|1)
; | | | | | | | | Motor Index Level (0|1)
; | | | | | | | | Index Mode
; | | | | | | | | Limit
; | | | | | | | | Overheat
; | | | | | | | | Index/Stall
; | | | | | | | | Error
```

In the case of open connectors, and no sensors attached, the CW and CCW limit levels should be programmed to 00 to prevent erroneous limit detection, and the Motor Index

level to 1 (a high) to keep the index count reset to prevent an erroneous index count overrun error.

Example:

To set the the Index level to high (1), the CCW level to 0, and the CW level to 0,

enter:

IF

The level byte is formed as follows:

$$\begin{aligned} \text{level} &= (4 * \text{Index Level}) + (2 * \text{CCW level}) + (1 * \text{CW Level}) \\ &= (4 * 1) + (2 * 0) + (1 * 0) = 4 \end{aligned}$$

Enter:

4 c/r

The unit echoes back ACK/NAK.

Index Left Limit.

Usage: IL

Returns ACK/NACK/EXCEPTION

To seek the LEFT or CCW limit, the motor is stepped at the at the low speed until the left position sensor is detected. This position is the "0" count position of the motor.

Before this command is executed, the motor speed and power settings must be programmed, as well as the index levels.

This command automatically resets the step mode to full step.

When the limit flag is detected during the move, the step operation in that direction will be terminated. For more details about limit programming see **Move Commands**.

Example:

To index the motor left enter:

IL c/r

At detection, the unit echoes back ACK/NAK.

Index Right Limit.

Usage: IR
Returns ACK/NACK/EXCEPTION

To seek the Right or CW limit, the motor is stepped CW at the at the low speed until the right position sensor is detected. This position is the "max" count position of the motor. This command should only be used after an **IL** commands which sets the "0" position

Before this command is executed, the motor speed and power settings must be programmed, as well as the index levels.

This command automatically resets the step mode to full step.

When the limit flag is detected during the move, the step operation in that direction will be terminated. For more detail about manual limit programming see **IH** and **IT** commands below.

Example:

To index the motor right enter:

IR c/r

At detection, the unit echoes back ACK/NAK.

Set Index Home

Usage: IH[Position]
where Position is a long value
At successful completion, the unit returns ACK
If failed, the unit returns NAK

Description:

This command sets the current position count to Position value. If you are not using limit detection sensors, set this value at any point less than or equal to the programmed maximum motor steps from left to right.

Default value is 2,147,483,648.

The left limit count is always 0. This command in combination with Set Maximum Travel (IT - see below) can be used to set a range of allowed travel if index flags are not available on the mechanism.

Note: if the right limit count is also programmed, and the value entered is less than the current value of the home position, the home position will be set equal to the right limit value. Use the IT command first to update the maximum travel limit before setting the current position.

Example:

To set the origin or home position enter:
IH

To set the home count to 8000 enter:
8000 c/r
The unit echoes back ACK/NAK.

Set Maximum Travel Index (Right Limit)

Usage: IM[Position]
where Position is a long value
At successful completion, the unit returns ACK
If failed, the unit returns NAK

Description:

After reset, the Left Limit count is set to 0, and the Right Limit step count is set to 4,294,967,295. If you are not using limit detection sensors, but want to set a maximum allowed right step count to a specific value, use this command to manually set the Right Limit step count value.

If the value of the current or home position is greater than the right limit count entered, the home position is set to the right limit value. Use this command first before the IH command to set the home position.

Example:

To set the right limit step count enter:

IM

To set the home count to 10000 enter:
10000c/r
The unit echoes back ACK/NAK/EXCEPTION.

Move Motor Commands

Move Absolute

Usage: MA[Position]
where Position is a long value
At successful completion, the unit returns ACK
If failed, the unit returns EXCEPTION

Description:

This commands steps the motor to the absolute step count of Position. If the position is beyond the right limit value, the unit will not move, but return **EXCEPTION**.

If an error is encountered (such as an unexpected limit detection flag), the unit will return **EXCEPTION**. In the event of an error, use the Read Status Command **RS** to determine what the error was. The controller is designed to terminate stepping immediately upon detection of a limit flag. If the unit was running at a high slew speed, final absolute step position cannot be guaranteed. The unit should be re-indexed for position accuracy after an error is detected.

The current stepping mode determines whether the physical step will be a full step or a fractional step. If the mode is set for half stepping, the sub-step register value will be updated.

Example:

To move to an absolute position,
enter:
MA [Position]c/r

If $0 \leq \text{Position} \leq \text{Right Limit}$,
The unit will calculate the number of steps and direction to step the motor until it reaches an absolute step count of Position. At completion of the move, if no errors are encountered, the unit returns **ACK**.

Move Relative

Usage: MR[{-}Offset]
where Offset is a long value
{-} is optional minus sign
At successful completion, the unit returns ACK
If failed, the unit returns EXCEPTION

Description:

This commands steps the motor to the step count of Position + {-}Offset. If the relative position is beyond the right or left limit value, unlike the MA command, the controller will step the motor up to the limit, then return **EXCEPTION**.

If an error is encountered while stepping, (such as an unexpected limit detection flag), the unit will return **EXCEPTION**. In the even of an error, use the Read Status Command **RS** to determine what the error was. The controller is designed to terminate stepping immediately upon detection of a limit flag. If the unit was running at a high slew speed, final absolute step position cannot be guaranteed. The unit should be re-indexed for position accuracy after an error is detected.

Example 2.3.1:

To move to an absolute position,
enter:

```
MR [{-}Offset]c/r
```

If $0 \leq \text{Position} + \text{Offset} \leq \text{Right Limit}$,

The unit will calculate the number of steps and direction to step the motor until it reaches a new position = Position+ Offset. At completion of the move, if no errors are encountered, the unit returns **ACK**.

Power Commands

Overview

The MB3001 uses the SLA7032 unipolar stepping motor controller chip. This chip is a Chopper Controller, it controls current (power) to the motor by allowing current to build up to programmable point, then switching to Pulse Width Modulation (PWM) to hold the current at that point.

Choppers are most efficient if the motor is driven at 2 to 4 times static voltage rating. For example, if the static rating of the motor is 6volt, a 24V drive will improve performance because it allows the drive current in the motor coils to build up at a faster rate.. For a discussion of operation see the MB3001 Hardware Manual.

The key consideration is of course, not to statically drive the motor at 24Volts, but to switch over to PWM when max current is reached to hold the drive current at safe level. The SLA7032 and DAC do this automatically.

The motor specification will either specify max current directly, or specify a motor coil resistance and voltage rating which can be used to calculate the max safe current.

CAUTION: Generally, overdriving a modern stepper motor will gain no benefit in performance, and result in damaging the motor by resistive overheating.

The current trip level is controlled by a 6bit Digital to Analog Converter (DAC). The motor current from each phase is sensed by a 1% 0.33 current sense resistor. The trip point is reached when:

$$V_{DAC} = I_{MOTOR} \times 0.33,$$

or:

$$I_{MOTOR} = V_{DAC} / 0.33$$

The DAC output is scaled so that at V_{MAX} value, I_{MAX} is approximately 1.5-A, so output current per bit of DAC accuracy is approximately.

$$1.5/256, \text{ or approximately } 6\text{-mA per bit}$$

Remember however, only the upper 6 bits of the DAC are significant, so only 64 levels are practically available, so the granularity of the DAC is actually:

$$6 * 4 = 24\text{-mA}$$

The above analysis assumes primarily a resistive load. The actual current levels at a specific trip point will vary according to the amount of inductance of the motor windings being switched. It should be noted that for motors with high resistance windings, this usually also means high inductance coils which can affect the effectiveness of the current chopping action of the controller.

For safe operation of the motor, use a RUN current value no greater than the static rating of the motor. It is not recommended to run at MAX current value, this may damage the controller chip

CAUTION: Even when operated within specification, the motor can be very hot! Some motors are designed to operate beyond 80°C - and can cause a severe burn if touched.

Set Hold Power Level

Usage: PH[Level]
where Level is a byte
Returns ACK

Description:

This command sets motor hold power level. Power levels are values from 0 to ff. Although the DAC uses a byte input value, the DAC is a 6-bit unit, the *lowest 2* bits are insignificant. Only 64 levels are practically available.

The Hold value is the level setting while the motor is idle. Usually the hold power can be set to 10% or less of the running power to minimize motor heating. See above for calculating the hold current value.

If the motor is not running, entering a value of 0 with the PH command will turn off the motor driver phases off completely for lowest power consumption. Next time the motor is moved, when it automatically powers down, the DAC is output is set for 0 current, BUT the phases are not turned off. This allows the motor to restart quickly without losing phase sync. Because of the way the SLA7032 chopper works, there is always some residual current even at a 0 setting if any of the phases are "on." If you want the phases totally off for lowest power drain, re-enter the PH command with a zero setting.

McGourty Associates, LLC

No matter what the setting, when motor hold current is reduced or off, there will be a slight change in the resting step position versus that when the motor is at full running current. This usually causes no problem, but it needs to be evaluated for the specific application at hand. It may be necessary to maintain a higher level of hold current to maintain position and acceptable motor holding torque.

Also, when starting to move from a stopped position, especially if under load, it may be necessary to power up the motor prior to moving.

Example:

To set the hold power levels enter:
PH

Enter 12 to set the hold current to 72-mA:
12 c/r

The controller will return ACK

Set Run Power Level

Usage: PR[Level]
where Level is a byte
Returns ACK

Description:

This command sets the motor run power level. Power levels are values from 0 to ff. Although the DAC uses a byte input value, the DAC is a 6bit unit, so lowest 2 bits are insignificant. Only 64 levels are practically available.

The Run value is the level setting while the motor is moving. Usually there is no performance advantage in setting the run current beyond the motor static rating. The running power is minimized to minimize motor heating. Because the PWM action of the controller works upon peak level detection, the RMS chopped output value may be less. This is highly dependent upon the motor inductance, resistance etc...

Example:

To set the run power levels enter:

PR

Enter 80 to set the run current to 0.48-A:

80c/r

The controller will return ACK

Read Data Commands

Read Analog Input

Usage: RA
Return: word

Description:

This command starts an an analog to digital conversion of the voltage at pin 2 of CN5, the analog input port. A 10bit value is returned. The value returned is proportional to VREF which is set to VCC (5VDC).

$$V_analog = VCC(data/1023)$$

Example:

To read an analog value at CN5, enter:
RA

the unit will return a word value equal
xxxx

The controller will return ACK

Read Driver Pins

Usage: RD
Returns byte

Description:

This command returns the current levels at the output [byte] to the SLA7032 for independent driver control. Only the lower nibble is returned.

The data returned is in binary format since control register data is typically bit oriented.

Example:

To read the current pin settings, enter:
RP

the unit will return the lower nibble of the byte value of the port contents - in binary format, the upper nibble is always 0000.

00000000 (or current value in binary)

followed by ACK.

Read Register/Port

Usage: RR[register]
where Port is a byte
Returns byte

Description:

This command returns the value at [register]. The value register, is a memory referenced value, 0 addresses register 0. I/O ports are offset by 20hex (32 decimal).

The data returned is in binary format since control register data is typically bit oriented.

Example:

To read a port, enter:

RR

Enter port address (e.g. 54 (0x36) to address PINB)

54c/r

the unit will return the byte value of the port contents - in binary format,
00000000 (or current value)

ollowed by ACK.

Read Status

Usage: RS
Return: Status Block (see below)

Description:

McGourty Associates, LLC

This command returns the unit ID, controller status word (as a 16 bit string of ASCII data), Current Position, and Right Limit Count (the Left Limit Count is always 0). All data is separated by commas.

The Status Block Data is formatted as follows:

```
unit_id:      .BYTE mb_u8      ;unit ID

motor_status: .BYTE mb_u16     ;motor status
;* status word format
;*
;* lo_byte   ; 7 6 5 4 3 2 1 0
;            ; 1 1 1 1 1 1 1 1
;            ; | | | | | | | | Direction CW=0:CCW=1
;            ; | | | | | | | | Run      0=stop, 1=run
;            ; | | | | | | | | Mode    0=full-step, 1=half-step
;            ; | | | | | | | | Mode2   2=reserved 3=reserved
;            ; | | | | | | | | AcDc    1=accel/decel enable
;            ; | | | | | | | | Limit1  signal state
;            ; | | | | | | | | Limit2  signal state
;            ; | | | | | | | | Index   signal state

;* hi_byte   ; f e d c b a 9 8
;            ; 1 1 1 1 1 1 1 1
;            ; | | | | | | | | Enable Hardware limits 1=set
;            ; | | | | | | | | CW Limit (0|1)
;            ; | | | | | | | | CCW Limit (0|1)
;            ; | | | | | | | | Index (0|1)
;            ; | | | | | | | | Limit
;            ; | | | | | | | |
;            ; | | | | | | | | Stall
;            ; | | | | | | | | Error

motor_status:      .BYTE mb_u16     ;motor status

:
:*****
;* system parameters
;*
position:  .BYTE mb_u32 ;current motor position
cw_limit:  .BYTE mb_u32 ;CW step limit
```

Example:

Assume that the unit is in terminal mode, the motor is at position 5000, and the CW limit is set to 10,000

To view current status,

Enter:

RS c/r

The controller returns:

255,0000000011100000,5000+0,10000

The unit will return ACK

All data is returned, even if in bus mode.

Read Version

Usage: RV

Returns ASCII string

Description:

This command returns the current software version of the control program. It is an ASCII string in the form of MB3001V1.0 060101A, where 060101A is the release date.

Example:

To read the current software version, enter:

RV c/r

The controller returns:

MB3001V1-00 060101A.

Speed Commands

Introduction

Usually a stepper motor is not operated at a single speed, but is programmed to start at a slow speed, then once moving accelerated to a faster or slow speed. The slow speed is selected to insure that starting and stopping will not result in any step position losses. Accurate step operation at the fast speed, requires an acceleration and deceleration ramp to run and stop without losing step accuracy.

There are three variables which set the operation performance, the Fast stepping Rate parameter, the Slow stepping Rate parameter, and the acceleration/deceleration Ramp. These parameters are *highly* system dependent - at what settings the system will reliably operate is dependent upon the motor constant, and system load characteristics.

Set Fast Stepping Rate

Usage: SF[interval]
where interval is an integer
Returns ACK/NAK

Description:

This command sets the step delay time for the `slew_rate`. The `slew` is the fastest rate the motor will run at without losing shaft position. Usually the motor cannot start and stop at this speed, but requires an acceleration and deceleration ramp to run and stop without losing step accuracy.

These values are used in conjunction with the acceleration/deceleration interval (see D command above). It is important to use values which divide evenly! Otherwise motor operation will be erratic.

Actual data entered is the step period = $1/\text{step_rate}$. The time base uses an interval of 0.5 μ sec. per digit. This gives a potential rate range of 0.0328sec to 0.5 μ sec.). In reality internal processing time will limit the upper rate to about 30kips. This is also typically beyond the speed capability of most common stepper motors.

If the `fast_rate` is slower than the `slow_rate`, it will be ignored, and the motor will run at the slow rate.

Enter the slow rate and ramp divisor first. When the the fast rate is entered, the ramp

interval is automatically re-calculated. If updating speed settings, update the slow rate and ramp values before entering the fast rate.

Example:

To set the fast rate enter:
SF

For a step rate of 4000pps, the delay required is 2.5×10^{-4} sec. The required delay count is: $2.5 \times 10^{-4} / 4 \times 10^{-6} = 62.5$, use 63 for a rate of 4032 pps.

Enter
63 c/r

The unit echoes back ACK/NAK

Set Accel/Decel Ramp Interval

Usage: SR[interval]
where interval is an integer
Returns ACK/NAK

Description:

At high speeds, or slewing, the torque of a stepping motor will decline, best motor torque is usually available only at low speed. Because of system inertia, a stepping motor cannot start from a stop and run instantly at a slew rate, or if running at a slew rate, cannot stop instantaneously without losing shaft positioning accuracy. To run with accuracy, to reach slew speeds, a stepping motor needs to be accelerated to slew speed, then decelerated to a stop. The MB3001 uses a trapezoid acceleration/deceleration profile. This profile consists of a linear acceleration ramp, followed by a plateau of constant velocity, ended by a linear deceleration ramp.

The SR command sets the number of steps for the acceleration/deceleration ramp. The acceleration/ deceleration curve is a linear ramp, where the integer value is the number of steps to take going from the low_rate to the slew_rate and back to the low rate before stopping. Acceleration and deceleration use the same profile (symmetrical).

In general, the more steps in the interval, the smoother the acceleration and deceleration, However, if the low speed is very low, excessive vibration may occur at the beginning of the

McGourty Associates, LLC

acceleration curve , and at the end of the deceleration curve. This is **very** system and motor dependent.

During acceleration the step delay (rate) at any point of the acceleration period is calculated as follows:

n = current step

$$\text{delay}^{n+1} = \text{delay}^n - (\text{low_rate} - \text{slew_rate}) / \text{acceleration steps}$$

During deceleration the step delay at any point of the deceleration period is calculated as follows:

n = current step

$$\text{delay}^{n+1} = \text{delay}^n + (\text{low_rate} - \text{slew_rate}) / \text{deceleration steps}$$

It is important to note that for processing speed, the Atmega8 CPU calculates the interval using binary division. It is important to select an interval that divides evenly into the difference between low speed – slew speed. Otherwise, the accumulated remainder or division error of an uneven division will cause a calculation error resulting in erratic performance.

Example:

To set a ramp interval of 64 steps (0x40 hex) enter **SR**

SR

Enter new value 64 and unit the unit will echo back the new setting

64 c/r

The unit will echo back ACK/NAK

Set Slow Stepping Rate

Usage: SS[interval]
where interval is an integer
Returns ACK/NAK

Description:

This command sets the step delay time for the slow rate. The slow is the non-accelerated rate the motor will run and stop without losing shaft step position.

The slow rate is the starting step rate of the motor during acceleration. The slow rate is also

the speed that the motor will use during left and right limit indexing.

Actual data entered is the step period = $1/\text{step_rate}$. The slow time base uses an interval of 4 μ sec. per digit.

When the the fast rate is entered, the ramp interval is automatically re-calculated, so if the slow rate is updated, the ramp interval will not be recalculated until the slow fast rate is updated.

Example:

To set the slow rate enter:
SS

For a step rate of 1000pps, the delay required is 1×10^{-3} sec. The required delay count is: $1 \times 10^{-3} / 4 \times 10^{-6} = 250$ pps.

Enter
250 c/r

The unit echoes back ACK/NAK

Set Step Mode

Usage: SM[mode]
where mode is a byte
Returns ACK/NAK

Description:

This command sets the stepping method. The MB3001 controller has two modes available, Full Step and Half Step.

Modes: 0 = full step
 1 = sub-step (half step)
 2 = wave-step
 3 = reserved

Step modes differences:

The most powerful mode of stepping is full step. Full stepping energizes two coils simultaneously to create each step phase.

Half stepping, uses a combination of a full step, and a half power (i.e. one phase on only) "in between" step. It increases the motor step resolution by a factor of two, but the torque of the in between half step is half that of a full step. At low speeds, because of the half power in between step, half stepping is a bit "softer" and quieter at each step transition. At slow speeds however, there is little advantage. If you want half step accuracy, but need to move a distance, for normal operation it may be better to run in full step mode - especially if slewing, then finish with a half step.

Before the advent of inexpensive microprocessors, wave step was a common drive method because it was easy to implement electronically. Wave step uses only one coil on at a time. It is the weakest drive of all the modes. There are some instances where the motor may run a bit quieter, if the reduced torque is acceptable.

Changing Step modes:

You can change between full and half stepping without losing step registration. However, a change to wave step from either full step or half step or back may lose step registration since the phase driving method is different.

Unit Commands

Set Unit ID

Usage: UI[ID]
ID is a byte value from 1 to 31 (0x01..0x1f).
Returns :ACK/NAK

Description:

This commands sets the unit address. The unit address is only required when operating in BUS mode. Each unit must have a unique ID

Example:

To set a new unit ID, enter
UI

Enter new [BYTE] (0..255)
BYTE c/r

and unit the unit will echo back
AC | NAK

Set Bus Mode

Usage: UB[bus}
If bus = 0, bus mode is set
if Bus = 255, terminal mode is set.

Description:

This commands sets the interface Bus mode. If mode = 255 (default), a terminal interface is selected, and conversation mode is enable. If the bus mode = 0, conversation mode is disabled and bus mode is selected.

Example:

To set the unit bus MODE, enter
UB

Enter new 255 c/r, to select Terminal mode,

McGourty Associates, LLC

All valid numbers will be accepted, Pressing the space key or carriage return key at any point will terminate the command with no changes made.

*If mode = 0, the unit will not echo back.

Because the RS232 and RS485 interfaces share the same SIO and require a manual jumper selection change, this command is primarily a debug command allowing MechaBus formatted data to be transmitted on the RS232C interface.

In terminal mode, all units will process the set bus mode command. To exit bus mode, a global address select **@00** must be sent first. The Set Bus mode will be received and processed by all units.

Restore Unit State

Usage: **UR** c/r
If success ACK is returned
if fail, NAK is returned

Description:

Before executing this command, a save state **US** command must be executed. This command restores the operation status to the values saved into back-up EEPROM with the save state command (see Save Unit State).

Prior to loading the backup data, the checksum created by the **US** command is verified. If the checksum passes, the data is loaded, If the checksums do not match, the operating state is not restored.

Example:

To restore the operating state
enter **UR**

UR c/r

If the calculated checksum matches the saved checksum, the checksum is returned, the state is restored and ACK

If the checksums do not match, NAK status is returned.

If the unit is in bus mode, ACK and NAK are not returned

Save status

Usage: **US**
Command returns ACK

Description:

This command saves the current operational state of the controller in EEPROM. Use this command after orderly stop BEFORE turning off the motor. When powered back up, use the UR command to restore the state of the controller.

When the data is saved, a checksum is calculated to insure accuracy during a Restore State Command (UR)

Example:

To save the operating state
enter:

US c/r

If the unit is in terminal mode, ACK is returned

Write Commands

Write Driver Pins

Usage: WD[byte]
Returns ACK/NAK

***** Caution! *****

- The SLA7032 is a low side or sink driver – A 0 bit value turns the driver on.
- Be careful not to exceed the absolute power rating of the device.
- For higher current drive, or AC power drive see the MB4001 Quad Relay Board.

Description:

This command outputs [byte] directly to the SLA7032 for independent driver control. Use this to drive independent relays, lamps, etc...

The SLA7032 is a sink driver. See the MB3001 Hardware manual for more information on how the SLA7032 chip functions.

There is a SLA7032 internal logic control restriction which allows only one driver of a pair on at a time. For example, InA and In/A may not both be on (low) at the same time. Setting them both on (low) will result in both outputs forced to off.

The A channel and B channel function independently of each other.

If you are using the MB3001 as a stepper motor controller, you should not ordinarily changes the pin driver values. If the port value is changed, and a step command is given, the controller may lose a couple of steps as it re-syncs the motor back into a proper phase sequence.

However, one potential use of the WD command is to move the motor a half-step after moving within a half of final position using full step mode (much faster operation). As long as a PH0 command was not given at the end of the move, you can read the current phase of the motor with a RD command. With that information you can manually output the next half step position. Starting from a half step to a full step in-the-same-direction, usually will cause no error. If you are reversing direction, depending upon loading of the system, you may need to manually re-establish the previous full step before executing the move.

Of course, if you are curious, you can manually step the motor outputting each step phase in sequence. Be sure to set the motor run power with a PR command before trying to move the motor.

In this case, after stepping, motor power will not be set automatically to the programmed hold level. You will need to do this manually with a PH command.

The driver pin port (PORTC) is mapped as shown below:

```

; 7 6 5 4 3 2 1 0
; x x x x 1 1 1 1
; | | | | | | | | InB
; | | | | | | | | In/B
; | | | | | | | | InA
; | | | | | | | | In/A
; | | | | | | | | x
; | | | | | | | | x these bits are ignored
; | | | | | | | | x
; | | | | | | | | x
```

Example

To turn on bit InB, enter:

WD

Enter 14 (0x0E) to turn INB on (active low input)

14c/r

The controller returns ACK

Modifying the Program

The Monitor and Dispatch routines

Since MechaBlox are designed for R&D use, it is expected that the end-user will want to make some modifications to the control software. For this reason, in the software design where ever possible, if the choice was between optimization or clearer understanding, the choice was made for clarity.

For ease of modification, MechaView software is constructed as a simple monitor loop driven by the serial input. Only the serial input receive routine is interrupt driven. The serial input routine receives ALL serial data on the bus, even if not addressed, and places it into a 16 byte FIFO (first-in-first-out) input buffer. With the exception of the emergency halt command "X", the receive routine processes nothing itself. Data processing is handled by the monitor routine.

The monitor routine reads the data as received from the FIFO input buffer maintained by the serial receive routine. The main purpose of the monitor is to "filter" the data so that only data specifically addressed to the unit is processed. Of course, in terminal mode, all data is processed.

First the monitor routine verifies that received data is a valid command - it must be alpha data. Valid commands data is A-Z, a-z and "@". The monitor routine automatically converts alpha data to upper case.

Actual command parsing is done by the dispatch routine. The dispatch routine is basically a jump table of control routine addresses. The alpha command is converted to jump table index address by the subtraction (command - '@'). The jump address read at the index is loaded into the Z register and an IJMP command is executed to the command control routine.

Adding Custom Commands

To add a custom command, select a letter command value not currently used (or re-map the values in the standard table), and add its control routine address to the jump table at the appropriate location.

Since the command is dispatched by a jump and not a call, to terminate the custom command and return control back to the monitor routine, use a return jump to one of the three standard return points:

returnACK = command completed successfully, returns '.'
returnNAK = command not understood '.'
returnException = command failed, returns '!'

These three points return control back to the monitor routine.

Before exiting the custom routine, be sure to enable interrupts if they were disabled during the routine. Otherwise, the serial interface routine will not service input data.

Programming examples

Initialization

Before the MB3001 unit can drive a stepper motor, the motor and system operating parameters must be set. At a minimum the motor slow, fast and ramp speeds, and power settings must be programmed. If system indexing signals are available, the controller must be programmed to read the proper index signal levels.

Step 1: Set motor speed parameters

SS1000 _{c/r}	Set slow speed to 1000
SR100 _{c/r}	Set acceleration / deceleration ramp to 100 steps
SF900 _{c/r}	Set fast speed to 900

Step 2: Set motor power parameters

PH10 _{c/r}	Set hold power to $(10 \times 6\text{ma}) = 60\text{mA}$
PR40 _{c/r}	Set run power to $(40 \times 6\text{ma}) = 240\text{mA}$

Step 3: Set index flag levels (optional)

IF4 _{c/r}	set the the shaft level to high (1), the CCW level to 0, and the CW level to 0 (default is 0 = shaft low (0), CCW low (0) and CW low (0))
--------------------	--

Step 4: Set Step Mode (optional)

SM1 _{c/r}	Set step mode to half step (if the motor is indexed using any of the index commands, this value will be changed back to full step, the default value)
--------------------	--

Step 5: Set current position Home value (optional)

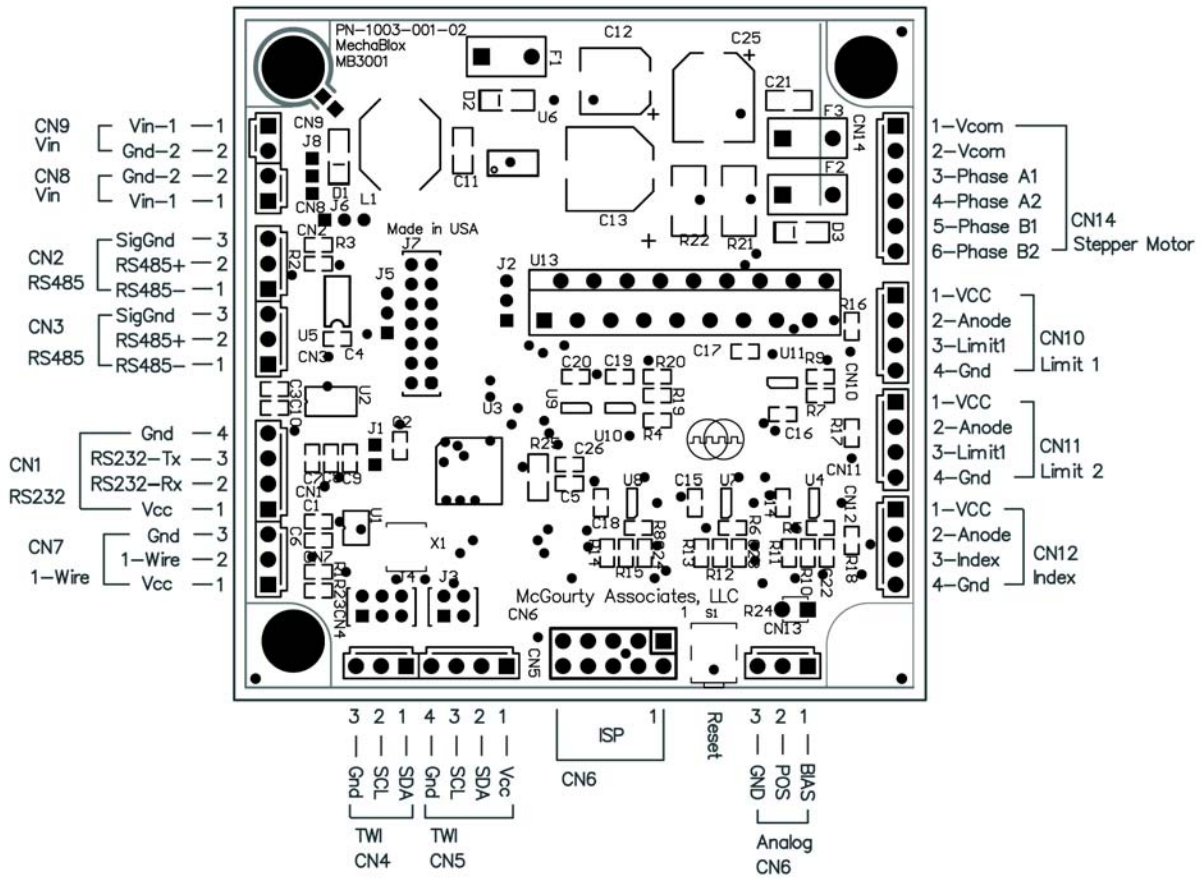
IH1000 _{c/r}	This commands set the current position as home position with a count value of 1000. At reset, the current location of the motor is the default home position at 2147483648.0, a very large value. Default left limit is always 0, and default right limit is 4,294,967,295.0
-----------------------	---

Step 6: Set right limit value (optional)

IT2000 _{c/r}	This commands set the maximum travel or right limit count to a value of 2000. At reset, the default right limit is 4,294,967,295. The left limit is always 0.
MR200 _{c/r}	This commands moves the motor CW 200 steps
MR-200 _{c/r}	This commands moves the motor CCW 200 steps, or back to original position
MA1000	

Board Setup

Connector Layout



MB3001 Connector Layout

Troubleshooting

<i>Symptom</i>	<i>Possible solution</i>
Unit does not respond to input	<ul style="list-style-type: none">• Check that DC power is connected and power supply is on.• Check that DC polarity is correct.• Check input power fuse.• Check proper COM cable is connected for communication mode selected.• Check J7 jumper settings are correct for mode selected• Check terminal baud rate and data setting
Unit sends garbage	<ul style="list-style-type: none">• Check proper COM cable is connected for communication mode selected.• Check terminal baud rate and data setting• Check J7 jumper setting are correct for communication mode selected.• If in bus mode, verify that all units are setup for bus mode
Motor moves, but in wrong direction	<ul style="list-style-type: none">• motor phases are reversed at connector, swap coil 1 pair at the connector
Motor will not move at all	<ul style="list-style-type: none">• Verify motor parameters are set-up correctly.• Check motor fuses.• Verify that hardware and software limit level settings are correct.
Motor hums and will not move at all	<ul style="list-style-type: none">• Motor speed is probably set too high, decrease pulse rate.

<i>Symptom</i>	<i>Possible solution</i>
	<ul style="list-style-type: none">• Motor run power level may be too low.
Motor moves erratically	<ul style="list-style-type: none">• Verify motor parameters are set-up correctly.• Check motor cable and connector.• Check motor fuses.• Motor run power level may be too low or too high for load inertia
Motor gets hot.	<ul style="list-style-type: none">• Decrease run and/or hold power settings
Motor loses step position	<ul style="list-style-type: none">• Motor speed settings are probably set too high, decrease pulse rate. Acceleration and Deceleration settings may be too short, or improper values, reprogram settings.• Hold and run power may be too low for load, increase power (do not exceed motor or MB3001 max values.
Limit Detectors do not work	<ul style="list-style-type: none">• Verify proper connection at connectors• Verify that sensor device type will work with the sensor interface circuit.• Verify proper level settings

Updating the Control Program

Where to find updates

MechaView is open source software and is distributed under the GPL software license (see below). The latest updates to the MB3001 control program can be found on the download page: www.mechablox.com/support/download.

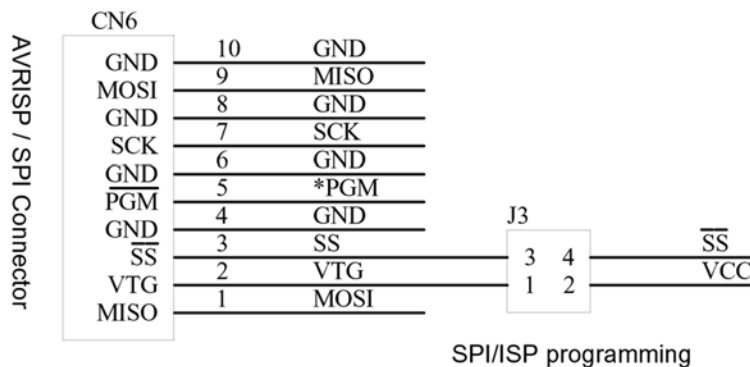
The software is developed under Atmel's, AvrStudio®. AvrStudio is not open source software, but an executable file can be freely downloaded at www.Atmel.com. Check the MechaView release note for which version of AvrStudio was used to assemble the release.

Using The AVRISP

The MB3001 controller supports the low-cost Atmel AVRISP in circuit programmer. There are several other compatible and open source versions of this device available. Get one and keep your hardware up to date!

Connector CN6 supports the 10pin IDC programming cable. Figure 3.5.3 from the MB3001 Hardware Manual is shown below. CN6 is wired ready for AVRISP operation, SS and VCC on Connector J3 are hardwired with traces to VTG and SS on CN6. (For special SPI applications ONLY, these traces can be cut, and a 2mm jumper installed for programming - usually this is NOT required)

Figure 3.5.3 ISP/SPI Interface



Disclaimer

McGourty Associates makes no warranty of any kind, expressed or implied, including any warranty of merchantability, fitness of the product for any particular purpose even if that purpose is known to McGourty Associates, or any warranty relating to patents, trademarks, copyrights or other intellectual property.

McGourty Associates shall not be liable for any injury, loss, damage, or loss of profits resulting from the handling or use of the product shipped.

McGourty Associates does not assume any responsibility for use of any circuitry described, no circuit patent licenses are implied.

McGourty Associates reserves the right at any time without notice to change product specifications as may be required to permit improvements in the performance, reliability, or restructurability.

McGOURTY ASSOCIATES' PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS.

Software License

CONDITIONS OF USE.

MechaView for MB3001 is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License (see below), or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

GNU GENERAL PUBLIC LICENSE

This document can be downloaded at
<http://www.gnu.org/licenses/gpl.txt>

GNU GENERAL PUBLIC LICENSE Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

McGourty Associates, LLC

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the

MECHAVIEW MB3001 Users Manual

Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

- a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
- b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
- c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of

McGourty Associates, LLC

a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

- a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and

all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

McGourty Associates, LLC

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.