

ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Visit our website - *Click HERE*

MSTEP-5

User Guide

KEITHLEY

User Guide

for the

MSTEP-5

Dual Stepper Motor

Controller Board

Revision B - April 1991
Copyright © Keithley MetraByte Corp. 1987
Part Number: 24892

KEITHLEY METRABYTE CORPORATION

440 MYLES STANDISH BLVD., Taunton, MA 02780
TEL. 508/880-3000, FAX 508/880-0179

Warranty Information

All products manufactured by Keithley MetraByte are warranted against defective materials and worksmanship for a period of one year from the date of delivery to the original purchaser. Any product that is found to be defective within the warranty period will, at the option of Keithley MetraByte, be repaired or replaced. This warranty does not apply to products damaged by improper use.

Warning

Keithley MetraByte assumes no liability for damages consequent to the use of this product. This product is not designed with components of a level of reliability suitable for use in life support or critical applications.

Disclaimer

Information furnished by Keithley MetraByte is believed to be accurate and reliable. However, the Keithley MetraByte Corporation assumes no responsibility for the use of such information nor for any infringements of patents or other rights of third parties that may result from its use. No license is granted by implication or otherwise under any patent rights of Keithley MetraByte Corporation.

Notes

MetraByte™ is a trademark of Keithley MetraByte Corporation.

Basic™ is a trademark of Dartmouth College.

IBM® is a registered trademark of International Business Machines Corporation.

PC, **XT**, **AT**, **PS/2**, and **Micro Channel Architecture**® are trademarks of International Business Machines Corporation.

Microsoft® is a registered trademark of Microsoft Corporation.

Turbo C® is a registered trademark of Borland International.

Contents

CHAPTER 1: INTRODUCTION

1.1 Description 1-1
1.2 General Areas Of Application 1-2
1.3 Utility Software 1-3
1.4 Technical Support 1-4
1.5 Options 1-4

CHAPTER 2: INSTALLATION

2.1 Backing Up The Disk 2-1
2.2 Hardware Installation 2-1

CHAPTER 3: PROGRAMMING

3.1 General 3-1
3.2 Register Locations & Functions 3-2
3.3 PPMC-101C Stepper Controller Internal Registers 3-5
3.4 Loading The Machine-Language Call Routine MSTEP.BIN 3-6
3.5 Structure Of The Call Statement 3-8
3.6 Error Codes 3-9
3.7 Stepper Motor Fundamentals 3-11
3.8 CALL Modes 3-12

CHAPTER 4: MODE CALLS

4.1 MODE 0: Emergency Stop 4-1
4.2 MODE 1: Decelerating Stop 4-2
4.3 MODE 2: Jog Or Single Step 4-2
4.4 MODE 3: Step With Acceleration/Deceleration 4-4
4.5 MODE 4: Step At Constant Speed 4-5
4.6 MODE 5: Move To Outer Limit At Constant Speed 4-7
4.7 MODE 6: Move To Limit At High Speed 4-9
4.8 MODE 7: Move To Base Point At Constant Speed 4-9
4.9 MODE 8: Read Motor Status 4-11
4.10 MODE 9: Read Encoder 4-14
4.11 MODE 10: Load Divider 4-16
4.12 MODE 11: Enable/Disable Interrupt 4-17
4.13 MODE 12: Initialization 4-19
4.14 Programmer's Short Summary 4-24
4.15 Interpreted BASIC (GW, Compaq, IBM, Etc.) 4-25
4.16 QuickBASIC 4-26
4.17 Multiple MSTEP-5s In One System 4-28

CHAPTER 5 ENCODER CHANNELS

5.1 Optical Incremental Shaft Encoder 5-1
5.2 MSTEP-5 Encoder Interface 5-2
5.3 Reading & Resetting The Encoder Counters 5-3
5.4 Connecting Up A Shaft Encoder 5-4

CHAPTER 6 USING THE SAT-STEP

6.1 STA-STEP Description 6-1
6.2 Power Drive Circuit 6-1
6.3 STA-STEP Connections To Stepper Motors 6-2
6.4 STA-STEP Connections To Encoders 6-2
6.5 Connection To External Translator 6-5
6.6 Other Signals On The STA-STEP 6-5

CHAPTER 7 STEPPER MOTORS & TRANSLATORS

7.1 How A Stepper Motor Works 7-1
7.2 Torque Versus Speed 7-3
7.3 Resonance 7-4
7.4 Full- & Half-Step Operation 7-4
7.5 Translators & Indexers 7-5
7.6 Mechanical Design 7-5

CHAPTER 8 TESTING & MAINTENANCE

CHAPTER 9 INSTRUCTIONS FOR PRODUCT RETURN

APPENDIX A PPMC-101C SPECIFICATIONS & PROGRAMMING

APPENDIX B SOURCES FOR STEPPER MOTORS & ASSEMBLIES

APPENDIX C STEP-MOT1 & STEP-ENC1 SPECIFICATIONS

APPENDIX D MSTEP-5 & STA-STEP SPECIFICATIONS

* * * * *

CHAPTER 1

INTRODUCTION

1.1 DESCRIPTION

MetraByte's MSTEP-5 is a plug-in 2-axis stepper motor and incremental shaft-encoder motion-control board for the IBM PC/XT/AT and compatible computers. The board is 12 inches long and requires a full-length expansion slot. All connections are made through the rear plate using a standard 50-pin insulation-displacement (mass-termination) connector. MetraByte also offers (as options) a screw connector, a stepper-motor-driver accessory (STA-STEP), a popular type of stepper motor (STEP-MOT1), and an incremental shaft encoder (STEP-ENC1; see Section 1.5). Figure 1-1 is a block diagram of the MSTEP-5.

Each independent stepper channel consists of a Sil-Walker PPMC-101C intelligent controller chip that can execute a variety of motion control commands. The PPMC-101C is one of the most popular Japanese robotics chips and has been designed to provide essential features with simplicity of use. MetraByte's driver software further enhances the ease of stepper motor control by personal computer.

Once a command has been loaded into the PPMC-101C controller chip, the host computer is no longer burdened by the execution of the particular motion, but may monitor its status as needed. The associated stepper motor may be moved any number of steps up to 24 bits of resolution (+/-16,777,216 steps) either with a controlled acceleration/deceleration profile or constant stepping rate. Associated with each motor are 5 limit switch inputs as well as a motor enabled input. The limit switches provide normal and emergency stop limits at both ends of travel, plus a home or reference point at any intermediate point. A normal stop is defined as a normal deceleration to rest without loss of the step count due to inertial effects, an emergency stop is a sudden stop that may lead to run on of the motor and hence loss of location from the step count and would normally require recalibration by return to the reference or home point. The emergency stop amounts to an immediate cessation of step pulses regardless of what the motor is doing at the time. In addition to controlling the number of steps travelled by the motor (normal motion), the PPMC-101C controller executes the following commands described below:

Initialization	Controls number of phases driven (3, 4 or 5), logic levels of phase excitation, (normal, inverted), internal/external step clock select and switching excitation on/off at standstill. In addition this command sets the start up, acceleration/deceleration and high speed run rates. The components of this command that relate to the motor configuration cannot be altered by a further initialization command without resetting the PPMC-101C controller.
Move Normal	Moves the motor the desired number of steps with a controlled rate and acceleration/deceleration.
Move Constant	Rotates motor at constant speed for a specified number of steps.
Find Limit	Rotates motor to an outer limit switch.
Find High-Speed Limit	Rotates motor to a high-speed or inner-limit switch.

Find Base Point	Rotates motor to home or reference-limit switch.
Read status	Read PPMC-101C controller status.
Decelerating Stop	Stops motor normally.
Emergency Stop	Instantly stops motor by removing drive pulses (may lead to loss of true location from step count).
Single Step	Single step or "jog" command.

Each stepper channel provides two different types of outputs. One is a counter-clockwise/clockwise (CCW/CW) signal plus a pulse-train output corresponding to the number of steps to be moved. This is suitable for driving a wide variety of standard stepper-motor translators available from most stepper-motor manufacturers. The other set of signals consists of 5-phase outputs used to drive power transistors to switch the stepper motor windings directly. The number of outputs enabled and the stepping sequence is controlled by the initialization command and can be matched to 3-, 4-, or 5-phase motors. MetraByte's STA-STEP accessory is an L/R type of driver including power mosfets driven by the phase outputs and can drive small motors directly. When the motor is at standstill and the winding excitation current is at a maximum causing the greatest heating, you can select a mode in the initialization that switches the phase outputs at 2.2KHz and about 30% duty cycle. This action takes place automatically about 100 milliseconds after standstill and reduces motor heating without much affecting holding torque. Also the logic polarity of all the phase drive outputs can be selected true or inverted.

The step rate is controlled by the clock frequency. The PPMC-101C chip has an internal clock of 12.5KHz plus a programmable divider that can divide by any number in the range 20-255 giving corresponding step rates between 49 to 625pps (pulses per second). This is usually adequate, but you also have the option of selecting an external clock. There are two external clock sources on the MSTEP-5, a 100KHz crystal, on-board clock with an 8-bit programmable divider, or an external user input. The additional 8-bit divider works in conjunction with the PPMC-101C internal divider to give step rates ranging from 1.54 to 5,000pps and further allows you to vary the step rate during operation or execution of a command. This lets you produce variable speeds with any motion command that is not directly possible with the internal PPMC-101C clock. The user input can provide even more flexibility. Selection of the clock source is software controlled.

In addition to the stepper-motor channels, there are two independent incremental shaft encoder channels. These may be ignored in open-loop stepper-motor control or used for closed-loop control or setting manual reference positions. The encoder channels are not in any way linked to the stepper-motor channels, so it is up to the host computer to intervene in any type of control utilizing the encoder inputs. Each encoder channel is TTL-compatible and consists of a 24-bit up/down counter for counting single edges with Channel A/Channel B quadrature-output type incremental encoders. The encoder counters may be cleared and read by the computer. The MSTEP-5 encoder inputs include a 2-stage shift-register digital filter to reject transient noise, this limits the encoder input pulse rate to 25KHz (about 1500 rpm with a 1000 line/rev. encoder). Cascade outputs are provided from the encoder counters to allow use of additional external counters if needed.

All communications with the MSTEP-5 is via I/O ports (no memory address space is used); eight contiguous addresses are used in I/O space. The Base Address is selected on a DIP switch and can be anywhere in the range 0 - 3F8 hex; 100-3F8 or 200-3F8 is the usual usable range in an IBM PC/AT/XT. More than one MSTEP-5 may be installed in a PC for multiple-axis control. The number is limited

only by available expansion slots.

The PPMC-101C stepper controller chips can also generate interrupts on completion of commands and certain other conditions. This is supported in the MSTEP-5 hardware, interrupts may be jumper selected to any of the IBM PC Interrupt Levels 2-7 and in conjunction with the PPMC-101C Status Registers makes interrupt handshaking a simple procedure and allows the programmer to perform background control.

1.2 GENERAL AREAS OF APPLICATION

Many manufacturers produce stepper motors, stepper motor assemblies such as X-Y tables, shaft encoders, etc. A short summary of sources appears in Appendix B. MetraByte can provide from stock a standard 5V, 1A, 200-step/revolution motor (STEP-MOT1) and 1,000-line/revolution optical shaft encoder (STEP-ENC1), see Appendix C for specifications. These devices are representative only of typical hardware that can be used with the MSTEP-5.

Generally, stepper motors are suited to applications requiring variable torque, low speeds (not exceeding 500 rpm) and maximum position retention. They also offer the simplicity of open-loop position control simply by counting the number of steps. If your needs dictate higher speeds, fast acceleration and deceleration, and the ability to return to a set position regardless of load perturbations, a closed-loop DC servo drive may be more suitable. MetraByte also offers this type of motion control board for the IBM PC (see our Model MSERVO-2 and accessories). This type of drive is generally more complex and costly than a stepper drive but has characteristics that may be essential in certain applications.

The MSTEP-5 may be combined with other MetraByte measurement and control boards to implement complex "move and measure" type of instrumentation. It has obvious applications in robotics, optics and lasers, mechanical assemblies, remote control, etc.

1.3 UTILITY SOFTWARE

It is possible to program the MSTEP-5 directly using normal I/O port commands (INP and OUT etc.). This is explained in Section 3 and Appendix A, but for the programmer who wants fast results, our accompanying utility software (MSTEP.BIN) will simplify the use of the MSTEP-5 and save a lot of programming time. The utility software is provided on a single-sided PC-DOS 1.10 format 5-1/4" floppy disk (upward compatible with DOS 2.0 and higher revisions):

1. A Microsoft Basic callable driver (MSTEP.BIN) is provided for control of the basic stepper and encoder functions. The fully commented assembly source for this driver (MSTEP.ASM) is also provided. The object module, MSTEP.OBJ, is also on the disk for linking when using compiled BASICs (for example, IBM BASIC Compiler, Microsoft QuickBASIC, etc.).
2. Examples and demonstration programs. A comprehensive demonstration program (DEMO.BAS, DEMO.EXE) is provided. This is excellent both as a programming example and a way of getting the "feel" of the PPMC motion commands and driver software features. It will also be useful in your system setup and test. For further details see Chapters 3 and 7.
3. Instructions for ASSEMBLY LANGUAGE, C, Pascal and Fortran programmers are included in Appendix E.

1.4 TECHNICAL SUPPORT

If you have a problem, need information or advice, give us a call [617-880-3000] and ask for applications engineering. We will do our best to assist you because our philosophy of doing business is to provide a solution to your problem, not just sell a piece of hardware. If for any reason you are dissatisfied with any MetraByte product or find it is unsuited to your requirements, you are welcome to return it within the first 30 days of purchase for a full refund. Please call us first for an R.M.A. (Return Material Authorization) number before sending back any hardware. The MSTEP-5 and accessories are warranted against defects in manufacture and material for 1 year from the date of original purchase.

1.5 OPTIONS

Several optional accessories to facilitate the use of the MSTEP-5 are available from MetraByte. These include:

1. STA-STEP: As an an aid to using the MSTEP-5, this combination screw terminal and power driver board accessory can be connected directly to the 50-pin rear connector of the MSTEP-5 using MetraByte's CACC-2000 cable. The STA-STEP accessory board performs two functions. All of the input and output connections of the MSTEP-5 (with the exception of the motor-phase drives) are brought out to miniature screw terminal connectors. The phase drives are optically isolated and drive ten Motorola MTPN-08 60v 25A 0.08 ohm power mosfet transistors (5 transistors for each of the 2 axes). Inductive clamping circuitry and provision for current limiting resistors are also provided on the board. For 3, 4 or 5 phase stepper motors up to about 24V 5A rating, all that is required is an external DC power source of appropriate voltage for the motors. Because of the opto-isolation, all of the motor power drive circuitry is isolated from the computer ground and MSTEP-5 outputs. This provides additional safety and convenience in application. See Chapter 5 for more details.

2. STEP-MOT1: MetraByte can provide (as a stock item) a Superior Electric stepper motor, type MO61-LS02 (or other manufacturer's equivalent). This has 200 steps per revolution (1.8 deg. step angle), 5v 1A excitation and 35 oz-in torque with an output shaft diameter of 0.25 inch. Specifications of this motor are provided in Appendix C . Other motor types should should be ordered direct from the manufacturer, a partial list of sources is in Appendix B.

3. STEP-ENC1: MetraByte can provide (as a stock item) an optical shaft encoder. This is 5v powered and outputs 1000 pulses/revolution. Additional specifications are provided in Appendix C.

If you wish to use a stepper motor translator card manufactured by another company, the MSTEP-5 pulse and CCW/CW outputs can be used to drive the translator. In this case, you can either make up your own cable, avoiding the use of the STA-STEP, or use the STA-STEP and simply pick up the desired connections on the miniature screw terminal connectors.

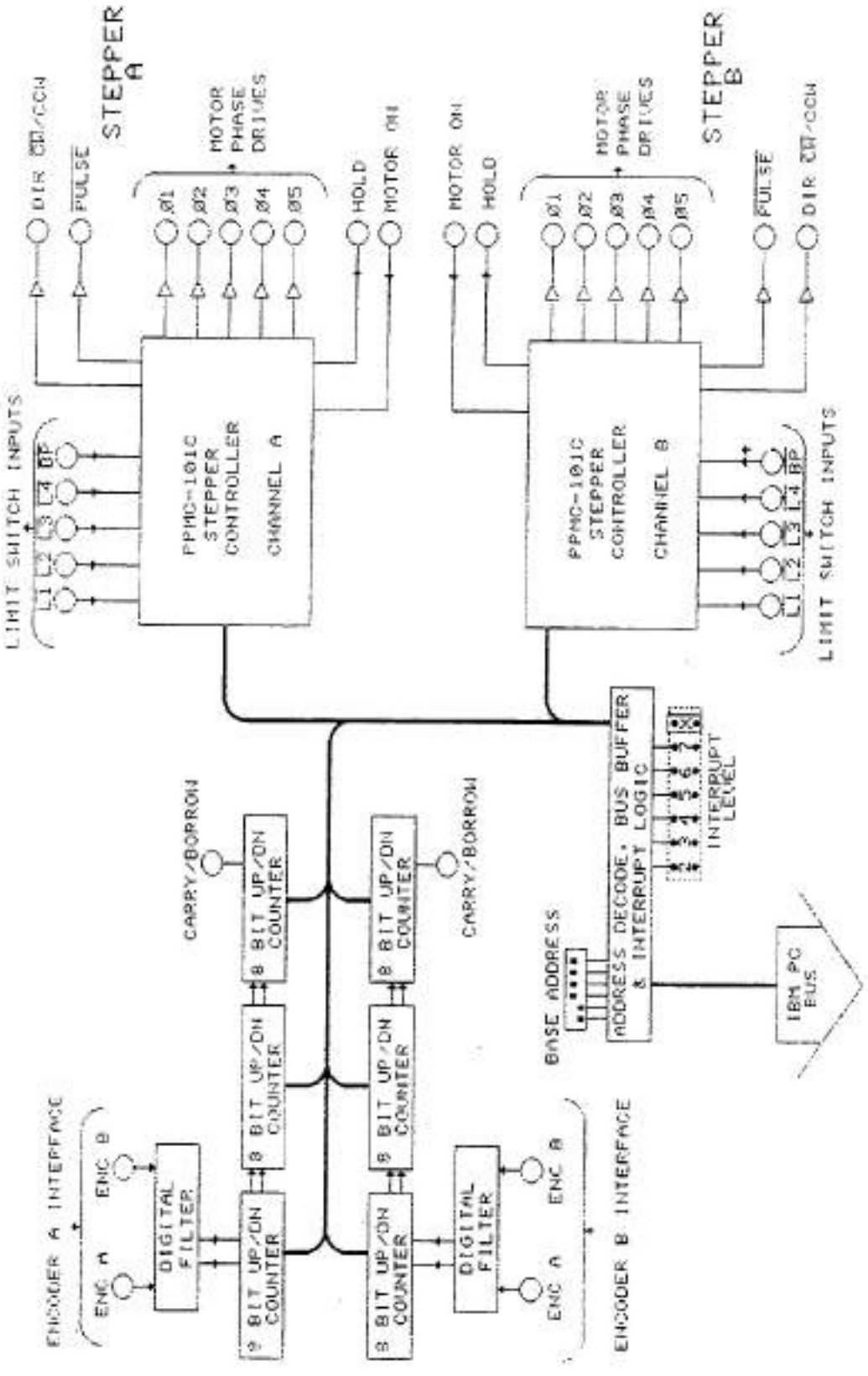


Fig. 1.1 BLOCK DIAGRAM OF MSTEP-5

* * * * *

CHAPTER 2

INSTALLATION

2.1 BACKING UP THE DISK

The Disk back-up utility software supplied with MSTEP-5 is in DOS 1.10 format and is compatible with DOS 2.0 and higher revisions. You are urged to make a back-up copy before using the software. For a direct back up, use the DOS DISKCOPY utility or alternatively COPY *.* to a pre-formatted disk. For a hard disk, simply use COPY *.* to transfer to a directory of your choice, the MSTEP-5 software is not copy protected. If for any reason you should misplace or destroy your MSTEP-5 utility software disk, please contact MetraByte for a free replacement.

2.2 HARDWARE INSTALLATION

MSTEP-5 utilizes 8 consecutive address locations in I/O space. Some I/O addresses will already be used by internal I/O and your other peripheral cards. To avoid conflict with these devices, MSTEP-5's I/O address can be set by the BASE ADDRESS DIP Switch to be on an 8-bit boundary anywhere in the IBM PC decoded I/O space. The IBM PC-XT I/O address space extends from decimal 512-1023 (Hex 200-3FF) and the IBM PC-AT I/O address space extends from decimal 256 to 1023 (Hex 100-3FF). In either case this is much larger than is ever likely to be fully occupied. Such a large space also allows use of more than one MSTEP-5 in a single computer. For your convenience, the reserved I/O addresses for standard IBM devices are detailed on the next page. You must avoid setting the MSTEP-5 to the same address as any other peripheral adapter card, but only if that other adapter card is already installed in your machine. A conflict of addresses will not cause physical damage but may cause malfunction of the MSTEP-5 and the conflicting adapter and in some circumstances, power on self test (POST) diagnostic messages. If you experience any of these problems, remove the MSTEP-5 and set it to another base address.

HEX RANGE	USAGE	HEX RANGE	USAGE
000 to 1FF	Internal System	387 to 37F	LPT1:
200 to 20F	Game	380 to 38C	SDLC comm.
210 to 217	Expansion unit	380 to 389	Binary comm. 2
220 to 24F	Reserved	3A0 to 3A9	Binary comm. 1
278 to 27F	Reserved	3B0 to 3BF	Mono dsp/LPT1:
2F0 to 2F7	LPT2:	3C0 to 3CF	Reserved
2F8 to 2FF	COM2:	3D0 to 3DF	Color graphics
300 to 31F	Prototype card	3E0 to 3E7	Reserved
320 to 32F	Hard disk	3F0 to 3F7	Floppy disk
		3F8 to 3FF	COM1:

The above list covers the standard IBM I/O options (most compatibles are identical), but if you have other I/O peripherals (special hard disk drives, special graphics boards, prototype cards etc.), they may be making use of I/O addresses not listed in the table above. Memory addressing is separate from I/O addressing so there is no possible conflict with any add-on memory that may be in your computer.

Usually, a good choices is to put the MSTEP-5 at Base Address Hex &H300 or &H310 (Decimal 768 or 784). (Note if you are using an IBM prototype board, it uses the Hex 300-31F address space and would

conflict, &H330 or &H340 would be a good alternative in this case). As an aid to setting the base address DIP switch, a graphical switch position display program *INSTALL.EXE* can be run from the DOS prompt. To run the installation aid program, type **A > INSTALL**.

When you get the `Desired base address?` prompt, type in your choice in decimal or IBM &H--- format and press **< Return >**. The program will round your address to the nearest 8-bit boundary, check for possible conflicts with standard IBM I/O devices (and warn you if so), and draw a picture of the correct positions of the seven toggles on the Base Address DIP switch. For additional details on Base Address switch settings, see Figure 2.0.

The next step is to remove the MSTEP-5 board from its protective electrostatic packaging and set the Base Address switch, located just to the upper left of the gold edge connector. It is a good precaution to discharge any electrostatic charge you may have accumulated by touching the metal frame of your computer before inserting the board in the computer.

The only other user setting is the choice of Interrupt Level. Most likely, you will not initially be making use of the interrupt capabilities of the MSTEP-5 and can park the "IRQ LEVEL" (interrupt level) selector jumper block in the "X" inactive position. If your programming will use interrupts from the MSTEP-5, then select the Hardware Interrupt Level 2 thru 7 that you intend to use. Take care to avoid selecting a level that is used by another adapter card (for example, Level 6 is always used by the floppy disk controller, Level 4 by COM1:, Level 3 by COM2: etc.). For more information on interrupt programming, see MODE 11, Section 3.8.12

To install the board, TURN OFF THE POWER on your computer and remove the case (See the IBM "Guide to Operations" manual if you are not already expert at this maneuver). Remove a vacant back plate by undoing the screw at the top and plug the MSTEP-5 in and secure the backplate. MSTEP-5 will fit in any of the regular full depth slots of the IBM PC-XT or Portable computer. On the PC-AT, it can be plugged into any socket but it will not make use of the extended AT bus interface connector. Due to the length of the 50-pin rear connector, it is advisable to set the retainer latches on this connector out straight, pass the connector through the rear slot and pivot the board down into the edge connector. The board is 1 inch shorter than the slot length to allow for this maneuver. Installation is now complete. You may plug any of the MSTEP-5 accessories or your own cable into the 50-pin connector on the rear.

Remember, TURN OFF THE POWER whenever installing or removing any peripheral board including the MSTEP-5. Never try to install or remove any peripheral board with the power on as it can cause costly damage to the electronics of your computer and/or the MSTEP-5 board.

If for any reason you later remove the MSTEP-5 board, MetraByte recommends that you retain the special electrostatically shielded packaging and use it for storage.

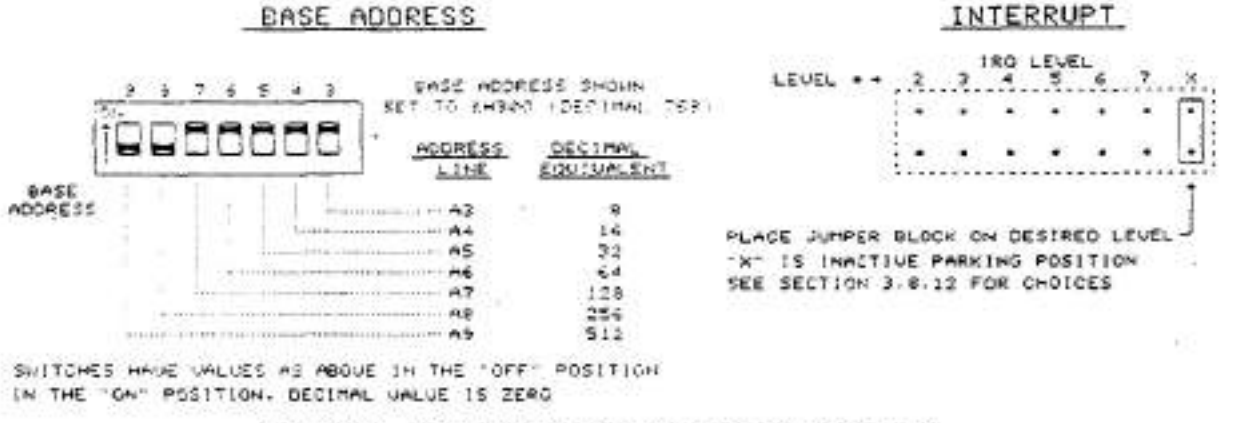


Figure 2.0. Base Address Switch & Interrupt Settings.

CHAPTER 3

PROGRAMMING

3.1 GENERAL

At the lowest level, the Stepper Motor Controller is programmed using I/O input and output instructions. In BASIC, these are the INP(X) and OUT X,Y functions. In Assembly Language, they are IN AL,DX & OUT DX,AL. Most other high level languages have equivalent instructions. Use of these functions usually involves formatting data and dealing with absolute I/O addresses. Although not demanding, this can require many lines of code and necessitates an understanding of the devices, data format, and architecture of the MSTEP-5. To simplify program generation, a special I/O driver routine "MSTEP.BIN" is included in the MSTEP-5 software package. This may be accessed from BASIC by a single line CALL statement. The CALL MSTEP will perform frequently used sequences of instructions. An example is Mode 1 which performs a normal stop. A routine to perform this operation in BASIC using INP's and OUT's would require many lines of code and would be rather slow and very tedious to program.

A sequence of BASIC INP's and OUT's to write the Stop Command to PPMC channel A would be as follows:

```
x10  CNR% = INP (BASADR% + 0)                    'read current control register
x20  CNR% = ((CNR% AND &HFC) OR &H05)            'select A, A0RSEL=1
x30  OUT BASADR + 0, CNR%                        'write control register
x40  STAT% = INP (BASADR% + 1)                   'fetch PPMC status
x50  IF (STAT% AND &H04) = 0 THEN GOTO xx150     'motor stopped?
x60  IF (STAT% AND &H02) <>0 THEN GOTO xx140     'input buffer full?
x70  CMD% = &H01 + &H10                          'stop Motor, disable interrupt
x80  CNR% = CNR% AND &HFE                        'set A0RSEL = 0
x90  OUT  BASADR% + 0, CNR%                      'write control register
x100 OUT  BASADR% + 1, CMD%                      'write PPMC command
x110 CNR% = CNR% OR &H01                         'set A0RSEL = 1
x120 OUT  BASADR% + 0, CNR%
x130 STAT% = INP(BASADR% + 1)                    'fetch PPMC Status
x140 IF (STAT% AND &H04) <> 0 THEN GOTO xxx130
x150 STOP                                        'finished
```

All this code can be circumvented by using the driver:

```
x10  MD% = 1                                     'command - decelerating stop
x20  D%(0)  =  1                                 'select channel A
x30  CALL MSTEP (MD%, D%(0), STP#, FLAG%)
```

Obviously, the MSTEP.BIN driver greatly reduces programming time and effort. Both methods of programming are described (see Appendix A for programming the PPMC-101C directly) and you are free to choose either. Usually the BASIC programmer will find the CALL routine method very much simpler to implement. The MSTEP.BIN driver also provides an example interrupt service routine. It is not possible to program interrupt routines directly in BASIC and the driver is the only way of utilizing interrupts from the MSTEP-5 hardware.

3.2 REGISTER LOCATIONS & FUNCTIONS

The I/O address map of the MSTEP-5 is structured as follows:

ADDRESS		READ	WRITE
Base Address	+ 0	Control Register (read/write register)	
	+1	PPMC Registers A/B (read/write registers)	
	+2	-	Clear Encoder A
	+3	Encoder Low Byte	Clear SMC A
	+4	Encoder Middle Byte	Clear SMC B
	+5	Encoder High Byte	Clear Encoder B
	+6	Clock Divider A	
	+7	Clock Divider B]	

Base Address +0: Control Register (Read/Write)

Access to each of the PPMC stepper controllers is through the I/O port at Base Address +1. The Control Register, at Base Address +0, acts as an indirect address or pointer register for the other devices on the board; it controls the following:

- Which of the PPMC stepper channels is accessed.
- Which of the internal registers (there are four in each controller) is selected.
- Which of the two encoder channels is read at Base Address +3, +4, or +5.
- Generation of interrupts.
- Choice of internal/external stepper-clock sources.

Bit functions in the control register are as follows:

BIT:	D7	D6	D5	D4	D3	D2	D1	D0
	XCB	XCA	EINTB	EINTA	ECTRAB	CSB	CSA	AORSEL

Bit functions:

XCB — Selects External Step Clock (XCB = 1) on Pin 42 or 100KHz + Divider on-board clock (XCB = 0) for stepper channel B. This applies only if the PPMC initialization command has specified external clock otherwise the 12.5KHz PPMC internal clock will be used.

XCA — Selects external step clock (XCA = 1) on pin 41 or 100KHz + divider on board clock (XCA = 0) for stepper channel A. This only applies if the PPMC initialization command has specified external clock otherwise the 12.5KHz PPMC internal clock will be used.

EINTB — Enables hardware interrupt from PPMC Stepper Channel B (EINTB = 1). Interrupt is disabled if EINTB = 0.

EINTA — Enables hardware interrupt from PPMC Stepper Channel A (EINTA = 1). Interrupt is disabled if EINTA = 0. If both EINTB and EINTB are enabled, interrupts from the 2 controllers are ORed together. In this case, the Interrupt Service Routine must poll each controller to determine the source of the interrupt.

ECTRAB Selects which of the Encoder Counter channels can be read at Base Address +3, +4, +5.

ECTRAB = 0: Encoder Channel A

ECTRAB = 1: Encoder Channel B

CSB, CSA, AORSEL select PPMC registers as follows:

CSB	CSA	AORSEL	READ(R)	WRITE(W)
1	0	0	PPMC Channel A Data Register (R/W)	
1	0	1	A Status Register	A Command Register
0	1	0	PPMC Channel B Data Register (R/W)	
0	1	1	B Status Register	B Command Register
0	0	0	Illegal	Write both data registers
0	0	1	"	Write both command registers
1	1	X	Unselected both channels	

Programming of the PPMC-101C controllers is covered in Section 3.3 and Appendix A.

Base Address +3, +4, +5: Encoder Counter Registers (Read Only)

Each Encoder Counter consists of three 8-bit sections that are read only. Channel A or Channel B is selected by the ECTRAB bit in the Control Register. The 24-bit Encoder Counter data is then read from Base Address +3, +4, and +5 as follows:

Base Address +3: Low byte (Read Only)
Base Address +4: Middle Byte (Read Only)
Base Address +5: High Byte (Read Only)

To clear Encoder Counter A, write to Base Address +2.

To clear Encoder Counter B, write to Base Address +5.

Note that the encoder data is not latched (the counter is read directly. To avoid erroneous data, the encoder data is best read when the encoder is stationary. Reading the counter with the encoder moving may give rise to errors in the data due to the delays between reading each byte, this depends to some extent on the speed of rotation of the encoder (see MODE 9, Section 3.8.10, and Section 4.3).

A typical encoder-read sequence in BASIC might be as follows:

```
x10 CNR% = INP(BASADR%)                    'Fetch current control register
x20 CNR% = ((CNR% AND &HF7) OR &H08)       '= encoder A, &H00=B
x30 OUT BASADR%, CNR%                      'set ECTRAB bit
x40 RDCL% = INP (BASADR% + 3)              'read low byte
x50 RDCM% = INP (BASADR% + 4)              'read middle byte
x60 RDCH% = INP (BASADR% + 5)              'read high byte
x70 POSN# = RDCL% + (256*RDCM%) + (65536*(RDCH% and &H7F))
x80 IF ((RDCH% AND &H80) <> 0) THEN POSN# = -POSN#
                                           'Position is CCW of initial point
```

This example illustrates a few important details. First the encoder channel to be read is selected with the ECTRAB bit in the Control Register. Since the data returned is 24-bit, it would ideally be suited to reading into a long (double-word) integer. Many languages (C, Fortran, etc.) provide long integers, but BASIC is an exception that forces you to use a floating point variable, instead. Note this needs to be a double-precision variable in BASIC to provide 24-bit resolution. Use a single precision variable and you lose counts at high values (to prove this, try X=17111113:PRINT USING "########";X). If we clear the encoder channel at the reference or "home" point of the system, the most significant bit of the up/down encoder counter can be treated as a sign bit and will indicate the rotation direction from the home point.

Base Address +2, +3, +4, +5: CLEAR Commands (Write Only)

Four *Clear* addresses have been provided. They are

1. A write to Base Address +2 clears Encoder Counter A.
2. A write to Base Address +5 clears Encoder Counter B.
3. A write to Base Address +3 resets PPMC A stepper controller.
4. A write to Base Address +4 resets PPMC B stepper controller.

A write to these registers at any time is similar to a hardware reset. The data written to these I/O addresses is irrelevant and is lost; it is the act of writing to the port that provides the *Clear* function. The PPMC-101C stepper controllers are specialized microprocessors that are automatically reset on power-up of the computer. Afterwards, these controllers can also be hardware-reset by a write to Base Address +3 or +4. After resetting a PPMC controller, **allow at least 100 micro-seconds to elapse before performing an initialize command**. This lets the specialized processor perform its own internal start-up initialization before responding to commands. In interpreted BASIC, allowing for this delay is not a concern, but in other languages, be sure to wait. An example of clearing all four registers in BASIC is as follows:

```
x10 OUT BASADR%+2, 0        'Clear Encoder A
x20 OUT BASADR%+5, 0        'Clear Encoder B
x30 OUT BASADR%+3, 0        'Clear PPMC A
x40 OUT BASADR%+4, 0        'Clear PPMC B
```

Base Address +6, +7: Clock Dividers (Write Only)

There are three different stepper-clock sources usable by the PPMC. PPMC internal 12.5KHz clock, Stepper Motor Controller Board external 100KHz clock, or an external user-supplied clock source. To use either of the clock sources external to the PPMC stepper controller chip(s), the PPMC initialization command must specify external clock input. Alternatively, the initialization command can also specify use of a 12.5KHz clock generated internal to the PPMC chip (internal clock), which is usually adequate for most applications. If the external clock option is selected on initialization, you have the further choice between external hardware inputs (Pins 41 and 42) or an on-board 100KHz crystal with programmable divider source. The external clock selection is controlled by the XCA and XCB bits of the Control Register.

If you elect to use the the Stepper Motor Controller Board external 100KHz Clock, you can select the pulse rate by loading the appropriate clock divider register. This register is Read/Write and may be programmed to select a pulse rate between 1.54 and 1,000 pulses per second (PPS). In addition, this register works in conjunction with the internal PPMC RA register to give an effective divider range of

20 to 65535.

The pulse frequency may be calculated using the formula:

PPS = 100,000 / ((FD+1) * RA)

where

FD = Frequency Divider (1 - 255)
RA = PPMC Divider (20 - 255)

If the PPMC Internal Clock and RA Divider are selected at initialization time, the pulse rate may be between 625 pps (RA = 20) and 49 pps (RA = 255). The formula to calculate this is

PPS = 12.5 / RA

where

RA = 20 - 255

If XCA or XCB = 1, a square-wave TTL-level clock may be provided on the external clock pins (Pins 41 and 42) as appropriate. To use this clock, the PPMC must be initialized for external clock. This clock source is divided by the internal PPMC RA register which is also set up at initialization time. The external input option allows user to synchronize the two stepper motor controllers with the same clock or change its rate during motion commands.

The pulse rate may be determined by the following formula:

PPS = (External Clock Rate (Hz)) / RA

where

RA = 20 - 255]

The following BASIC code is an example of programming the MSTEP-5 board clock dividers (an identical function is performed by MODE 10 of the driver):

```
xxx10 CNR% = INP(BASADR% + 0)      'read current control register
xxx20 CNR% = CNR% AND &H3F         'turn off XCA & XCB - select 100KHz
xxx30 OUT BASADR%+0, CNR%          'write control register
xxx40 OUT BASADR%+6, DIVA%         'write Divider A
xxx50 OUT BASADR%+7, DIVB%         'write Divider B
```

3.3 PPMC-101C STEPPER CONTROLLER INTERNAL REGISTERS

Each PPMC-101C Stepper Controller is a specialized microprocessor. The MSTEP-5 Control Register selects the PPMC channel (A or B) through the CSA and CSB bits, and any of the four internal PPMC registers by the AORSEL bit:

AORSEL	READ	WRITE
0	DATA	DATA
1	STATUS	COMMAND

Motion commands are issued to the PPMC controller by writing a command code to the Command Register (with AORSEL = 1). Since the PPMC may be busy executing a command, you must always read the STATUS register to determine whether the PPMC is ready to receive the command. After a command is issued, it may require or produce a variable number of data bytes (depending on the command) which are written or read from the Data Registers (with AORSEL = 0). Apart from controlling access to the Command Register, the Status register also provides additional information on the operation of the PPMC controller.

There are eight motion-control commands as well as the INITIALIZATION command that set the operating conditions of the controller. The functions of the Command and Status Registers are as follows:

INITIALIZATION	Selects the motor type, method of excitation acceleration/deceleration rate, internal/ external step clock, phase output logic type and start up and high speed pulse rate. Once the initialization command has been sent, it may not be overwritten by a further initialization command. The PPMC controller must be reset either by turning the computer power off and on or more conveniently by issuing a hardware clear command (see Sections 3.2.4 and 3.8.13 - mode 12).
OPERATION	This is the user interface mode in which COMMAND selects any of eight motion-control commands. The length of the data to follow depends on the specific command.
STATUS	Before/after the completion of an operation command, the status register provides data on the limit switches, motor in motion or at standstill and input/output data buffer full or valid. It also enables you to read the number of steps remaining to be travelled etc.

Direct programming of the PPMC-101C controllers and their full specification is more fully covered in Appendix A.

3.4 LOADING THE MACHINE-LANGUAGE CALL ROUTINE MSTEP.BIN

In order to make use of the CALL routine MSTEP.BIN, it must first be loaded into memory. You must avoid loading it over any part of memory that is being used by the main body of your program, DOS, or programs such as RAM disks that use high memory. If you do collide with another program, your computer will usually hang up although sometimes the results can be more peculiar. Often you will need to turn the power off and restore it to re-boot the machine, the usual Ctrl-Alt-Delete reset may fail to restore DOS. This may sound ominous, but apart from the frustration, no damage will ever result!

MSTEP.BIN uses about 3 Kbytes of memory and is best loaded outside BASIC's workspace. A typical loading sequence is as follows:

```
xx100 DEF SEG = &H3000        'segment of memory to load link
                              (choose an empty area e.g. @@ 192K)
xx110 BLOAD "MSTEP.BIN",0     'load driver  . .  Continue program
```

The above initializing steps will be the same for any interpreted BASIC program. A more comprehensive example is provided on the disk in DEMO.BAS. Note that the DEF SEG = &H3000 statement in line 100 specifies the load location for the MSTEP.BIN driver. All subsequent CALL's will occur to the last DEF SEG address, so if you add other DEF SEG's in your program, remember to precede your CALL's to MSTEP-5 with the same DEF SEG = &H3000 that you used to load the link (see CALL and DEF SEG in your BASIC reference manual).

Finding a place to load MSTEP.BIN is seldom much of a problem now that most PC's are equipped with at least 256K of memory. The following explanation provides some insight into the process of choosing a memory location for the driver and what to do if memory is in short supply.

DOS occupies the bottom of memory, the amount of memory required being dependent on the version (it grows as each new revision adds extra features!). The simplified memory map below shows what happens after booting up BASICA.

	DOS 1.1		DOS 2.1		DOS 3.0
Bottom:	0K ------	0K	-------	0K	------
	DOS				
	19K ------				
			DOS		
					DOS
		47K	-------		
	BASIC			63K	------
			BASIC		
					BASIC
	98K -------				
	Free memory	126K	-------		
			Free memory	140K	------
					Free memory

MSTEP.BIN should be loaded somewhere in the free memory area so that it does not interfere with either BASIC or DOS. This would be above 98K (&H1880) for DOS 1.1, 126K (&H1F80) for DOS 2.1 or 140K (&H2300) for DOS 3.0. If you have 256K (&H4000) or more of memory, then loading the link at DEF SEG = &H2800 or &H3000 is a good solution for all versions of DOS. One further small detail is that if you are using a PC compatible which does not have BASIC in ROM like the IBM PC, then BASIC (e.g. GWBASIC) is usually loaded as an .EXE file from the top of memory down. This is likely to fill up to 64K of the top segment of memory. Some virtual disks or print spoolers will do the same. Also if you are accustomed to using DOS resident programs such as Borland's Sidekick etc. be aware that these will push the loading floor of BASIC up and require a compensating increase in the location of MSTEP.BIN.

If you are memory limited, or you have so much resident stuff that there is no longer 64K left for BASIC to load in, then BASIC will attempt to make the most of what it can find. Instead of getting the message when BASIC has loaded:

```
The IBM Personal Computer Basic
Version A2.0 Copyright IBM Corp. 1981, 1982, 1983
60865 Bytes free
OK
```

You may get only 49345 bytes free (or something less than 60000 bytes) for example. In this case make a note of what space BASIC has found. You can then contract this space further using the CLEAR function and load the link at the end of BASIC. This is more complicated, but just as effective.

Let's suppose we get the message 52000 bytes free. MSTEP.BIN will use 3K bytes, so to be on the safe side let's force BASIC to use 48K. Our initializing code would now be:

```
xx100 CLEAR, 48000          'contracts BASIC workspace
```

Next we need to find out where BASIC has loaded in memory, add 48000 to it and load MSTEP.BIN just after the end of BASIC workspace. Memory locations &H510 and &H511 always contain BASIC's load segment:

```
xx110 DEF SEG = 0                          'set up to read &H510 and &H511
x120 LS = 256*PEEK(&H511)+PEEK(&H510)      'load segment
x130 SG = LS + 48000/16                    'remember segment addresses are on
                                                 16-byte (paragraph) boundaries
x140 DEF SEG = SG                          'set up to load link
x150 BLOAD "MSTEP.BIN",0                   'load link
  . .
Proceed with your program as before
```

3.5 STRUCTURE OF THE CALL STATEMENT

If you are new to using CALL statements, this explanation may assist you in understanding how the CALL transfers execution to the machine language (binary) driver routine (also see CALL in your Basic Reference Manual). Prior to entering the CALL, the DEF SEG = SG statement sets the segment address at which the CALL subroutine is located. The CALL statement for the MSTEP.BIN driver must be of the form

```
xxxx CALL MSTEP (MD%, D%(0), STP#, FLAG%)
```

Let us examine the parameters after CALL one by one:

MSTEP — In interpreted BASIC this is a variable that specifies the offset of the start of our routine from the segment defined in the last DEF SEG statement. In our case its value is always set to zero (MSTEP = 0). In compiled BASIC (and most other compiled languages) MSTEP has a different significance - it is the name of the external routine that the linker will look for. Note: We would have liked to use the name STEP instead - it's a better mnemonic than MSTEP, but be warned that STEP is a reserved word (as in FOR I=0 TO 6 STEP 2) and CALL STEP would produce a syntax error.

MD% This is an integer variable that specifies the operation that we wish the driver to perform e.g. MD%=0 performs an emergency stop, MD% = 12 initializes a channel etc. In the case of this driver, valid mode numbers range from 0 to 12.

D%(9) This is a 10-element integer array that passes data to and from the driver. The signifigance of particular data items varies according to the mode (MD%) selected. Not all elements of D%(*) are used in all modes.

STP# This is a double precision variable that specifies the direction and number of steps to travel or returns optical shaft encoder counts. The sign indicates the direction, + clockwise, - counter-clockwise. Not all modes utilize the STP# data , however it must always be included in the call parameter list.

FLAG% Returns an error code if any of the specifying D%(*) or MD% are out of range or if the motor is busy or at standstill in certain commands. In the case of no error, FLAG% is returned zero.

The four variables within brackets are known as the CALL parameters. On executing the CALL, the addresses of the variables (pointers) are passed in the sequence written to BASIC's stack. The CALL routine unloads these pointers from the stack and uses them to locate the variables in BASIC's data space so data can be exchanged with them. Three important format requirements must be met, as follows:

1. The CALL parameters are positional. The subroutine knows nothing of the names of the variables, just their locations from the order of their pointers on the stack. If you write:

   ```
   xxxxx CALL MSTEP (D%(0), FLAG%, MD%, STP#)
   ```

   you will mix up the CALL routine, since it will interpret D%(0) as the mode data, and FLAG% as the D%(0) data variable etc. The parameters must always be written in the correct order, as follows: mode #, data, step count, errors

2. The CALL routine expects its parameters to be of correct type and will write and read to the variables on this assumption: integer, integer array, double precision, integer

   If you slip up and use the wrong variable types in the CALL parameters, the routine will not function correctly and may hang up the program.

3. You **cannot** perform any arithmetic functions within the parameter list brackets of the CALL statement. There can only be a list of variables. Also you are not allowed to replace variables by constants.

Apart from these restrictions, you can name the variables what you want, the names in the examples are just convenient mnemonics. You should always declare the variables before executing the CALL so that BASIC has reserved memory locations for them before entering the CALL. In the case of the integer array, the first element D%(0) should be specified in the CALL parameter list as the data variable so that the CALL routine can locate all of the other remaining data items in the array correctly.

3.6 ERROR CODES

Some value checking is performed on entry data and any errors discovered are returned in FLAG%. This is primarily to prevent you setting up the CALL with obviously incorrect data such as interrupt level 9, mode number -6, base address 2000, byte output data 299 etc. and is intended to help avoid a bad setup of the hardware which could hang the computer. Also certain commands, such as an

emergency or decelerating stop are redundant if the motor is already at standstill (FLAG% = 7) and the PPMC controller may not be receptive to further commands if it is already busy executing a command (FLAG% = 1). If a non-zero error code is returned in any mode, execution of that mode will have been abandoned without action since error checking precedes any I/O to the hardware.

ERROR CODE #	PROBLEM
0	No error, OK
1	Motor busy
2	Driver not initialized on Channel A
3	Driver not initialized on Channel B
4	Mode number <0 or >12
5	Hardware error
6	Step count out of range +/-16,777,215
7	Motor already at standstill
8	- (not used)
9	-
10	Error in range of D%(0)
11	Error in range of D%(1)
12	Error in range of D%(2)
13	Error in range of D%(3)
14	Error in range of D%(4)
15	Error in range of D%(5)
16	Error in range of D%(6)
17	Error in range of D%(7)
18	Error in range of D%(8)
19	Error in range of D%(9)

Checking for errors is easily performed after each CALL and is recommended if not as a permanent feature then at least while debugging your program:

```
xxx00 CALL MSTEP (MD%, D%(0), STP#, FLAG%)
xx10 IF FLAG%<>0 THEN PRINT "Error number ";FLAG%:STOP
```

Certain error codes are useful in performing chained motion commands, as follows:

```
xxx00 MD% = 3                              'accelerate/decelerate command
xxx10 STP# = 1000                          'steps & direction to move
xxx20 D%(0) = 1                            'select channel B
xxx30 CALL MSTEP (MD%, D%(0), STP#, FLAG%)
xxx40 IF FLAG%<>0 THEN PRINT "Error number ";FLAG%:STOP
xxx50 STP# = -99                           'now do 99 steps in opposite
                                               direction
xxx60 CALL MSTEP (MD%, D%(0), STP#, FLAG%)
xxx70 IF FLAG% = 1 THEN GOTO xxx60         'loop while motor busy
xxx40 IF FLAG%<>0 THEN PRINT "Error number ";FLAG%:STOP
```

Note that after issuing a motion command in line xxx30 the controller and motor will still be busy executing this command when the program reaches the next motion command in line xxx60. In this case, FLAG% will return 1 and we can keep trying to execute line xxx60 until the previous command of line xxx30 has finished. In this way we can execute a whole series of commands as fast as the hardware will allow.]

3.7 STEPPER MOTOR FUNDAMENTALS

In order to program a stepper motor channel, it is necessary to understand a little bit about the physical setup and characteristics of a stepper motor. The motor itself consists of a permanent magnet rotor rotating within a multipole stator. As the current is switched in the windings of the stator, the magnetic field advances from one pole to the next and pulls the rotor along with it (see Section 6 for more information). The rotor can be turned a precise amount simply by controlling the number of energization/de-energization cycles of the windings (steps).

On turning on the power, the position of the motor is usually unknown, so it is necessary to move it to some known reference position called the home, base or reference point. When the mechanical system is in this position it can operate a switch (microswitch, hall effect or optical interrupter) and this provides the BP (base point) limit switch input. Once at the base point, all further commands can move relative to this known location, and as long as there is no loss of power or emergency stops the subsequent position can always be determined from the step count.

In practical systems, there is usually some physical limit on how far the motor can turn e.g. a lead screw gets wound to an end. To stop the motor, there is provision for 4 sets of limit switches, L1 - L4. L3 and L4 provide inner or deceleration limits where it is desirable to stop a fast moving motor even though it has not reached the absolute physical limit of travel. L1 and L2 are end of travel limits (or emergency stop limits) where it is essential to stop the motor immediately. A typical physical representation of a system is shown in Figure 3-1.

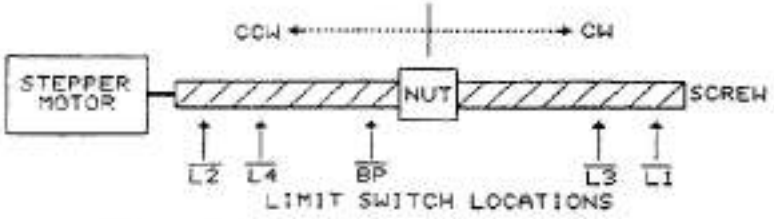


Figure 3-1. Typical mechanical arrangement.

Note that a mechanical system can be simplified and does not have to have 5 limit points, although the PPMC-101C controller can handle it. For example, L2 & L4 & CNP may be one physical limit switch, and L1 & L3 another for a 2 limit switch system. If you do this, you may lose some performance features of the PPMC, but gain in mechanical simplicity. Simply common up the limit switch inputs as required.

A mechanical system has significant inertia and it is not usually possible to start or stop the motor abruptly. The PPMC-101C controllers look after this problem by starting up the motor at a slow rate, ramping it up to full speed over a prescribed number of steps, and slowing it back down again before reaching the final step count (see Figure 3-2). All these parameters are set in the initialization command (mode 12). The inner limit switches (if used) allow you to perform a controlled stop without loss of positional count before hitting the outside or emergency stop limits. Whenever the motor hits an emergency limit switch, L1 or L2, during a motion command, it will stop. Which limit switch it hit and how many pulses remain to be executed in the last command and what was the last motion command can all be determined from a status read (mode 8). In this way the controller prevents overtravel and damage to the mechanical system and the programmer can provide appropriate corrective action depending on which of the limit switches were activated.

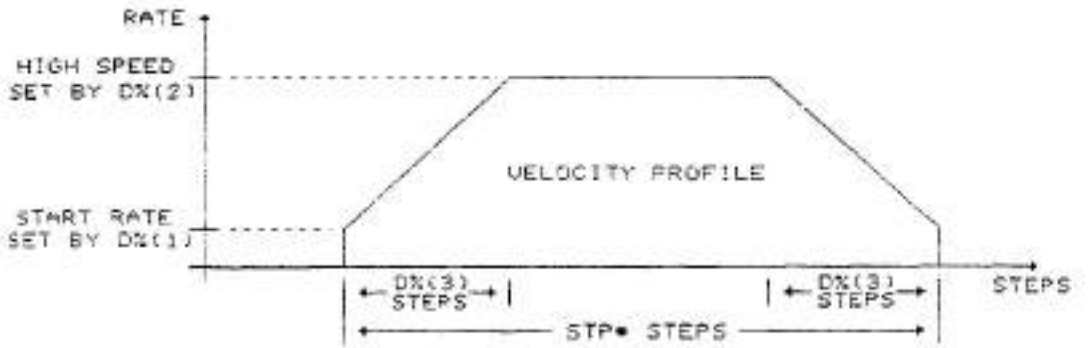


Figure 3-2. Acceleration/deceleration profile (MODE 3).

3.8 CALL MODES

The MSTEP.BIN driver supports thirteen different modes (numbered 0 thru 12). For simplicity, MODEs 0 thru 7 correspond identically to the command numbers in the PPMC-101C data sheet (see Appendix A). The additional MODEs 8 thru 12 support other functions. Each mode performs a specific operation as described below:

MD%	= 0	Emergency stop
	= 1	Decelerating stop
	= 2	Jog (single step)
	= 3	Accelerate/decelerate STP# steps
	= 4	Travel STP# steps at constant speed
	= 5	Move to a normal limit at constant speed
	= 6	Move to overtravel limit at high speed
	= 7	Move to base point at constant speed
	= 8	Read motor status
	= 9	Read encoder position
	= 10	Load external clock divider
	= 11	Enable/disable interrupts
	= 12	Initialize, reset & clear encoders

The method of use of each mode and the significance of D%(*) is described fully in the following subsections. **It is essential to perform a channel initialization (MODE 12) separately on each channel before other MODEs 0 - 11 may be selected.**

* * * * *

CHAPTER 4

MODE CALLS

4.1 MODE 0: EMERGENCY STOP

MODE 0 performs an emergency stop or immediate cessation of driving pulses to a motor that is busy executing a motion command. On receipt of the command, inertia may cause a rotating motor to run on, so the step count may no longer be an accurate guide to the motor position. An emergency stop would usually be followed by a re-calibration or return to home position.

Entry Data:

MD% = 0	Emergency stop command
D%(0)	Selects Channel A (0) or Channel B (1)
D%(1) thru (9)	Value irrelevant
STP#	Value irrelevant
FLAG%	Value irrelevant

Exit Data:

D%(0) thru (9)	Unchanged
STP#	Unchanged
FLAG%	0 if executed OK 2 if driver not initialized on Channel A 3 if driver not initialized on Channel B 4 if MD% <0 or >12 5 if hardware error 7 if motor already at standstill (command redundant) 10 if D%(0) not 0 or 1

A typical program entry preceding MODE 0 might read as follows:

```
xxx10 MD% = 0                                  'select MODE number
xxx20 D%(0) = 1                                'select Channel B
xxx30 CALL MSTEP (MD%, D%(0), STP#, FLAG%)
                                               'emergency stop
xxx40 IF (FLAG%<>0 AND FLAG%<>7) THEN PRINT "Error";FLAG%;" in MODE 0":STOP
xxx50                                          'continue program
```

One detail concerning the preceding example: If FLAG% = 7 is returned, it is just a reminder that you told the controller to do an emergency stop when in fact the motor was already stopped. In this case the driver aborts the command and lets you know why through FLAG%.

4.2 MODE 1: DECELERATING STOP

MODE 1 performs a decelerating stop or gradual cessation of driving pulses on a motor that is busy executing a motion command. The deceleration parameters correspond to those set in initialization MODE 12. This type of stop should lead to no loss of positional accuracy of the motor. The number of pulses which the motor would have continued to step if it had not received a decelerating stop command can be determined by a status read (MODE 8). The exact position of the motor can then be determined and a recovery routine initiated if required.

Entry Data:

MD% = 1	Decelerating stop command
D%(0)	Selects Channel A (0) or Channel B (1)
D%(1) thru (9)	Value irrelevant
STP#	Value irrelevant
FLAG%	Value irrelevant

Exit Data:

D%(0) thru (9)	Unchanged
STP#	Unchanged
FLAG%	0 if executed OK 2 if driver not initialized on Channel A 3 if driver not initialized on Channel B 4 if MD% <0 or >12 5 if hardware error 7 if motor already at standstill (command redundant) 10 if D%(0) not 0 or 1

A typical program entry preceding MODE 1 might read as follows:

```
10 MD% = 1                    'select MODE number
20 D%(0) = 0                        'select Channel A
30 CALL MSTEP (MD%, D%(0), STP#, FLAG%) 'decelerating stop
40 IF (FLAG%<>0 AND FLAG%<>7) THEN PRINT "Error";FLAG%;" in MODE 1":STOP
50                                  'continue program
```

One detail concerning the preceding example: If FLAG% = 7 is returned, it is just a reminder that you told the controller to do a decelerating stop when in fact the motor was already stopped. In this case the driver aborts the command and lets you know why through FLAG%.

4.3 MODE 2: JOG OR SINGLE STEP

MODE 2 performs a single step or "jog" of the stepper motor useful in getting a system into final position or for manual control. The direction is set by the sign of the STP# data although the value does not matter. In this respect STP# = 0,1,99 or 16,123,678 would all produce a 1 step clockwise

motion, whereas STP# = -1, -234, or -13,456,987 would all produce a 1-step counter-clockwise motion.

Entry Data:

MD% = 2	Single step (jog) command
D%(0)	Selects Channel A (0) or Channel B (1)
D%(1) thru (9)	Value irrelevant
STP#	Sign sets direction, magnitude irrelevant
FLAG%	Value irrelevant

Exit Data:

D%(0) thru (9)	Unchanged
STP#	Unchanged
FLAG%	0 if executed OK 1 if motor already busy executing last command 2 if driver not initialized on Channel A 3 if driver not initialized on Channel B4 if MD% <0 or >12 5 if hardware error 10 if D%(0) not 0 or 1

A typical program entry preceding MODE 2 might read as follows:

```
xxx10 MD% = 2                                  'select MODE number
xxx20 D%(0) = 0                                'select Channel A
xxx30 CALL MSTEP (MD%, D%(0), STP#, FLAG%)
                                               'jog command
xxx40 IF FLAG%<>0 THEN PRINT "Error in MODE 2, # ";FLAG%:STOP
xxx50                                          'continue program
```

Here's a programming tip. This routine uses the cursor keys to jog clockwise or counterclockwise until another key is pressed:

```
100 MD% = 2                              'select MODE
110 D%(0) = 1                                  'select Channel B
120 A$ = INKEY$:IF A$="" GOTO 120              'wait for keypress
130 IF LEN(A$)=1 THEN GOTO 210                 'exit on any single key code
140 IF ASC(RIGHT$(A$,1))=75 THEN STP#=-1:GOTO 170
                                               'cursor left
150 IF ASC(RIGHT$(A$,1))=77 THEN STP#=+1:GOTO 170
                                               'cursor right
160 GOTO 210                                   'exit on any other double key code
170 CALL MSTEP (MD%,D%(0),STP#,FLAG%)          'jog
180 IF FLAG%=1 THEN GOTO 170                   'repeat command if still busy
190 IF FLAG%<>0 THEN ?"Error in jog command":STOP
200 GOTO 120                                   'read keyboard again
210                                            'continue your program here
```

4.4 MODE 3: STEP WITH ACCELERATION/DECELERATION

MODE 3 is the basic all purpose trapezoidal motion command that causes the motor to move the number of steps specified by STP# with acceleration, deceleration, start up and high speed run rates set by the initialization parameters of MODE 12. The direction of rotation is controlled by the sign of STP#, positive is clockwise and negative is counter-clockwise. Up to 16,777,215 steps may be performed with one command corresponding to the 24 bit integer data limits of the PPMC-101C controllers.

MODE 3 corresponds to motion command 3 of the PPMC-101C with a few "user friendly" differences introduced by the driver. The PPMC will in fact move 1 step more than the number input, the driver corrects this characteristic by subtracting 1 from the step count before inputting it to the PPMC, so that STP# corresponds exactly with the number of steps moved. Also the PPMC behaves in strange ways with step counts of -1,0 and +1. Again the driver intercepts these singular values, does nothing with a STP#=0 (aborts command with no error) and automatically reverts to a jog command if STP# = +1 or -1 (see Appendix A for further details). This is all transparent to the user, so you can compute values of STP# without worrying about the controller's peculiarities. In fact, you can ignore MODE 2 and do jogs with STP# = +1 or -1 if this rationalization makes sense for you.

If the the total number of steps is less than twice the acceleration/deceleration step count, the velocity profile will be triangular instead of trapezoidal i.e. the motor will never reach its high speed run rate. The PPMC controller will look after this automatically.

Entry Data:

MD% = 3	Accelerate/decelerate command
D%(0)	Selects Channel A (0) or Channel B (1)
D%(1) thru (9)	Value irrelevant
STP#	Value and sign set number of steps and direction
FLAG%	Value irrelevant

Exit Data:

D%(0) thru (9)	Unchanged
STP#	Unchanged
FLAG%	0 if executed OK 1 if motor already busy executing last command 2 if driver not initialized on Channel A 3 if driver not initialized on Channel B 4 if MD% <0 or >12 5 if hardware error 6 if STP# <-16,777,215 or >+16,777,215 10 if D%(0) not 0 or 1

A typical program entry preceding MODE 3 might read as follows:

```
xxx10 MD% = 3                    'select MODE number
xxx20 D%(0) = 1                  'select Channel B
xxx30 STP# = -5632               'move counter-clockwise 5,632 steps
```

```
xxx30 CALL MSTEP (MD%, D%(0), STP#, FLAG%)
                                          'acel/decel command
xxx40 IF FLAG%<>0 THEN PRINT "Error in MODE 3, # ";FLAG%:STOP
xxx50                                     'continue program
```

Here's a programming tip. The time this command takes to complete depends on the stepping rate and number of steps, for a lot of steps it can take a long time. If the command is reissued while a previous motion is taking place the FLAG% will be returned = 1, indicating that you have to wait before sending the next motion command (unless it's a stop command). Let's say we would like the motor to move 1,237 steps clockwise, 67 steps counter-clockwise, 12,678 steps clockwise etc. This is how to program this trajectory:

```
100 MD% = 3                                   'select MODE
110 D%(0) = 1                                 'select Channel B
120 STP# = 1237                               '1st step count
130 CALL MSTEP (MD%, D%(0), STP#, FLAG%)
                                              '1st motion
140 IF FLAG%=1 THEN GOTO 130                  'keep trying if motor busy
150 IF FLAG%<>0 THEN "Error # ";FLAG%:STOP
                                              'major disaster!
160 STP# = -67                                '2nd step count
170 CALL MSTEP (MD%, D%(0), STP#, FLAG%)
                                              '2nd motion
180 IF FLAG%=1 THEN GOTO 170                  'keep trying until 1st motion
                                                   finished
190 IF FLAG%<>0 THEN "Error # ";FLAG%:STOP
                                              'major disaster!
200 STP# = 12678                              '3rd step count
210 CALL MSTEP (MD%, D%(0), STP#, FLAG%)
                                              '3rd motion
220 IF FLAG%=1 THEN GOTO 210                  'keep trying until 2nd motion
                                                   finished
230 IF FLAG%<>0 THEN "Error # ";FLAG%:STOP    'major disaster!
240 'etc.
```

Obviously a good programmer would use a few GOSUBs to minimize the code above. Also note that the program is paced by the rate at which the motor will execute commands. Maybe you would like to go off and do something else if you find FLAG%=1, the option is yours. Also if you ever need to find whether the motor is busy without doing anything before inputting a motion command, use read status, MODE 8 - see MODE 8 for further details.

4.5 MODE 4: STEP AT CONSTANT SPEED

MODE 4 is similar to MODE 3 except that the motor will move the number of steps specified by STP# with constant velocity specified by D%(1). The stepping rate depends on the clock source. For the PPMC internal clock source (D%(7) = 0 in initializing MODE 12),

Rate = 12,500 / D%(1) steps/second

For the external user input clock source (D%(7) = 2 in initializing MODE 12),

Rate = Ext. Freq. / D%(1) steps/second

For the on board 100KHz clock + divider (D%(7)=1 in initializing MODE 12),

Rate = 100,000 / (D%(1) * (X+1)) steps/second

where X = divider ratio set in MODE 10

The valid range for D%(1) is 20 (fastest) to 255 (slowest). The direction of rotation is set by the sign of STP#, positive is clockwise and negative is counter-clockwise. Up to 16,777,215 steps may be performed with one command corresponding to the 24 bit integer data limits of the PPMC-101C controllers.

Since the motor has to start up and run at a constant rate, this rate must be within the start up capabilities of the motor. Also, the rate set in this MODE by D%(1) **does not overwrite** the rate specified for acceleration/deceleration in initializing MODE 12 i.e. it is specific to this command only.

If you are operating from the on-board 100KHz clock and external divider, it is possible to change the external divider ratio while the motor is in motion and in this way modulate the speed. This is a MODE of operation that is not obtainable with the PPMC controller alone, at least not with one continuous command.

MODE 4 corresponds to motion command 4 of the PPMC-101C with a few "user friendly" differences introduced by the driver. The PPMC will in fact move 1 step more than the number input, the driver corrects this characteristic by subtracting 1 from the step count before inputting it to the PPMC, so that STP# corresponds exactly with the number of steps moved. Also the PPMC behaves in strange ways with step counts of -1,0 and +1. Again the driver intercepts these singular values, does nothing with a STP#=0 (aborts command with no error) and automatically reverts to a jog command if STP# = +1 or -1 (see Appendix A for further details). This is all transparent to the user, so you can compute values of STP# without worrying about the controller's peculiarities. In fact, you can ignore MODE 2 and do jogs with STP# = +1 or -1 if this rationalization makes sense for you.

Entry Data:

MD% = 4	Move at constant speed command
D%(0)	Selects Channel A (0) or Channel B (1)
D%(1)	Sets speed, valid range 20 - 255
D%(2) thru (9)	Value irrelevant
STP#	Value and sign set number of steps and direction
FLAG%	Value irrelevant

Exit Data:

D%(0) thru (9)	Unchanged
STP#	Unchanged

FLAG%	0 if executed OK 1 if motor already busy executing last command 2 if driver not initialized on Channel A 3 if driver not initialized on Channel B 4 if MD% <0 or >12 5 if hardware error 6 if STP# <-16,777,215 or >+16,777,215 10 if D%(0) not 0 or 1 11 if D%(1) <20 or >255

A typical program entry preceding MODE 4 might read as follows:

```
xxx10 MD% = 4                                        'select MODE number
xxx20 D%(0) = 1                                      'select Channel B
xxx30 D%(1) = 180                                    'speed, about 70pps with
                                                          internal clock
xxx40 STP# = 2000                                    'move clockwise 2,000 steps
xxx50 CALL MSTEP (MD%, D%(0), STP#, FLAG%)
                                                     'constant speed command
xxx60 IF FLAG%<>0 THEN PRINT "Error in MODE 4, # ";FLAG%:STOP
xxx70                                                'continue program
```

As with MODE 3, you can chain motion commands using FLAG%=1 to signal the readiness of the controller to receive the next command (see MODE 3 for programming example).

4.6 MODE 5: MOVE TO OUTER LIMIT AT CONSTANT SPEED

MODE 5 lets you run the motor into either of the outer overtravel limit switches L1 or L2. The motor will move in the direction specified by the sign of STP# with constant velocity specified by D%(1) until it encounters the appropriate limit switch input (L1 for clockwise, L2 for counter-clockwise). The stepping rate depends on the clock source. For the PPMC internal clock source (D%(7) = 0 in initializing MODE 12),

Rate = 12,500 / D%(1) steps/second

For the external user input clock source (D%(7) = 2 in initializing MODE 12),

Rate = Ext. Freq. / D%(1) steps/second

For the on board 100KHz clock + divider, (D%(7)=1 in initializing MODE 12),

Rate = 100,000 / (D%(1) * (X+1)) steps/second

where X = divider ratio set in MODE 10.

The valid range for D%(1) is 20 (fastest) to 255 (slowest). Since the motor has to start up and run at a constant rate, this rate must be within the start up capabilities of the motor. Also, the rate set in this MODE by D%(1) **does not overwrite** the rate specified for acceleration/deceleration in initializing MODE 12 i.e. it is specific to this command only.

If you are operating from the on-board 100KHz clock and external divider, it is possible to change the external divider ratio while the motor is in motion and in this way modulate the speed.

Entry Data:

MD% = 5	Move at constant speed to outer limit L1 or L2.
D%(0)	Selects Channel A (0) or Channel B (1)
D%(1)	Sets speed, valid range 20 - 255
D%(2) thru (9)	Value irrelevant
STP#	Sign sets direction, magnitude irrelevant
FLAG%	Value irrelevant

Exit Data:

D%(0) thru (9)	Unchanged
STP#	Unchanged
FLAG%	0 if executed OK, otherwise:- 1 if motor already busy executing last command 2 if driver not initialized on Channel A 3 if driver not initialized on Channel B 4 if MD% <0 or >12 5 if hardware error 6 if STP# <-16,777,215 or >+16,777,215 10 if D%(0) not 0 or 1 11 if D%(1) <20 or >255

A typical program entry preceding MODE 5 might read as follows:

```
xxx10 MD% = 5                                    'select MODE number
xxx20 D%(0) = 0                                  'select Channel A
xxx30 D%(1) = 250                                'speed, about 50pps with
                                                       internal clock
xxx40 STP# = -99                                 'move counter-clockwise to L2
xxx50 CALL MSTEP (MD%, D%(0), STP#, FLAG%)
                                                 'move to limit L1 or L2
xxx60 IF FLAG%<>0 THEN PRINT "Error in MODE 5, # ";FLAG%:STOP
xxx70                                          'continue program
```

After you have reached the limit, you can confirm that you are there by performing a MODE 8, read status command. In general, the constant speed should be selected slow enough so that there is no danger of overshooting the limit. It is then possible to use either L1 or L2 as homing reference points as an alternative to the BP (base point) or CNP input.

4.7 MODE 6: MOVE TO LIMIT AT HIGH SPEED

MODE 6 lets you run the motor with controlled acceleration and deceleration to and slightly beyond either of the inner high speed limit switches L3 or L4. The motor will move in the direction specified by the sign of STP# with a trapezoidal velocity profile specified by the initialization parameters of MODE 12. On hitting the appropriate inner or high speed limit switch (L3 clockwise or L4 counter-clockwise) a controlled deceleration will take place, and hopefully (if you have your system set up right!) will stop before reaching an outer limit switch L1 or L2. At this point you can inch it into an outer limit using MODE 5 to perform a calibration, or do whatever else is appropriate in the circumstances. This provides a fast way of getting to the end of travel if it is something you need to do frequently.

Entry Data:

MD% = 6	Move at constant speed to inner limit L3 or L4.
D%(0)	Selects Channel A (0) or Channel B (1)
D%(1) thru (9)	Value irrelevant
STP#	Sign sets direction, magnitude irrelevant
FLAG%	Value irrelevant

Exit Data:

D%(0) thru (9)	Unchanged
STP#	Unchanged
FLAG%	0 if executed OK 1 if motor already busy executing last command 2 if driver not initialized on Channel A 3 if driver not initialized on Channel B 4 if MD% <0 or >12 5 if hardware error 6 if STP# <-16,777,215 or >+16,777,215 10 if D%(0) not 0 or 1

A typical program entry preceding MODE 6 might read as follows:

```
xxx10 MD% = 6                                        'select MODE number
xxx20 D%(0) = 0                                      'select Channel A
xxx30 STP# = -8                                      'move counter-clockwise to L4
xxx40 CALL MSTEP (MD%, D%(0), STP#, FLAG%)
                                                     'move to limit L3 or L4
xxx50 IF FLAG%<>0 THEN PRINT "Error in MODE 6, # ";FLAG%:STOP
xxx60                                                'continue program
```

4.8 MODE 7: MOVE TO BASE POINT AT CONSTANT SPEED

MODE 7 lets you run the motor into the BASE point limit switch (this is called CNP on the PPMC data sheet) at constant speed set by D%(1) and direction set by the sign of STP#. The magnitude of STP# is irrelevant.

Not all systems will use a base point reference switch, often one of the outer limits L1 or L2 will do just as well to calibrate the motor position. If however you do provide a base point reference, say in the middle of travel, how do you know which way to go to find it? The answer is that you don't, if you have gone the wrong way you may just as likely stop at one of the outer limit switches. The PPMC is not intelligent enough to reverse direction on hitting an end limit, so you had better be prepared to provide some software to do the hunting! A status read, MODE 8, after MODE 7 will tell you what limit you hit, BP or an outer limit, L1 or L2. If you hit an outer limit, you can engage MODE 7 again in the reverse direction and you are bound to find the BP this time - see MODE 8 for details.

The stepping rate in MODE 7 depends on the clock source. For the PPMC internal clock source (D%(7) = 0 in initializing MODE 12),

Rate = 12,500 / D%(1) steps/second

For the external user input clock source (D%(7) = 2 in initializing MODE 12),

Rate = Ext. Freq. / D%(1) steps/second

For the on board 100KHz clock + divider, (D%(7)=1 in initializing MODE 12),

Rate = 100,000 / (D%(1) * (X+1)) steps/second

where X = divider ratio set in MODE 10

The valid range for D%(1) is 20 (fastest) to 255 (slowest). Since the motor has to start up and run at a constant rate, this rate must be within the start up capabilities of the motor. Also, the rate set in this MODE by D%(1) **does not overwrite** the rate specified for acceleration/deceleration in initializing MODE 12 i.e. it is specific to this command only.

If you are operating from the on-board 100KHz clock and external divider, it is possible to change the external divider ratio while the motor is in motion and in this way modulate the speed.

Entry Data:

MD% = 7	Move at constant speed to base point BP or CNP
D%(0)	Selects Channel A (0) or Channel B (1)
D%(1)	Sets speed, valid range 20 - 255
D%(2) thru (9)	Value irrelevant
STP#	Sign sets direction, magnitude irrelevant
FLAG%	Value irrelevant

Exit Data:

D%(0) thru (9)	Unchanged
STP#	Unchanged

FLAG%	0 if executed OK 1 if motor already busy executing last command 2 if driver not initialized on Channel A 3 if driver not initialized on Channel B 4 if MD% <0 or >12 5 if hardware error 6 if STP# <-16,777,215 or >+16,777,215 10 if D%(0) not 0 or 1 11 if D%(1) <20 or >255

A typical program entry preceding MODE 7 might read as follows:

```
xxx10 MD% = 7                                    'select MODE number
xxx20 D%(0) = 0                                  'select Channel A
xxx30 D%(1) = 200                                'speed, about 62pps with
internal clock
xxx40 STP# = 10                                  'move clockwise to BP
xxx50 CALL MSTEP (MD%, D%(0), STP#, FLAG%)
                                                 'move to BP
xxx60 IF FLAG%<>0 THEN PRINT "Error in MODE 7, # ";FLAG%:STOP
xxx70                                        'continue program
```

4.9 MODE 8: READ MOTOR STATUS

MODE 8 provides useful information about what the motor is doing or has done. The PPMC-101C controller can only provide full status information if the motor is at rest. If the motor is busy, FLAG% will be returned = 1 and the status data variables D%(2) thru D%(4) will be returned unchanged. This at least tells you that the motor is busy and unable to respond to a motion command other than an emergency or decelerating stop. At this point you can decide whether to let the motor complete whatever its doing and continue looping status reads until FLAG%=0 indicating its finished, or you can intervene and abort whatever it's doing with a stop command.

If FLAG% is returned = 0 then D%(2) thru D%(4) and STP# will contain status data as follows:

D%(2) contains the **FINISH STATUS** consisting of a byte of data:

```
B7   B6   B5   B4   B3   B2   B1   B0
 |    |    |    |    |         |
 |    |    |    |    |      Last command code 0-7
 |    |    |    |    |
 |    |    |    |   Decelerating stop on L3 or L4
 |    |    |    |
 |    |    |   Stop on limit L1 or L2
 |    |    |
 |    |   Motor on (MC) check flag
 |    |
 |   Stopped by a motor stop command
 |
Finish flag or interrupt flag
```

Bits B0 thru B2 provide the code or MODE number of the last command. Bits B3 thru B7 tell you what happened as follows:

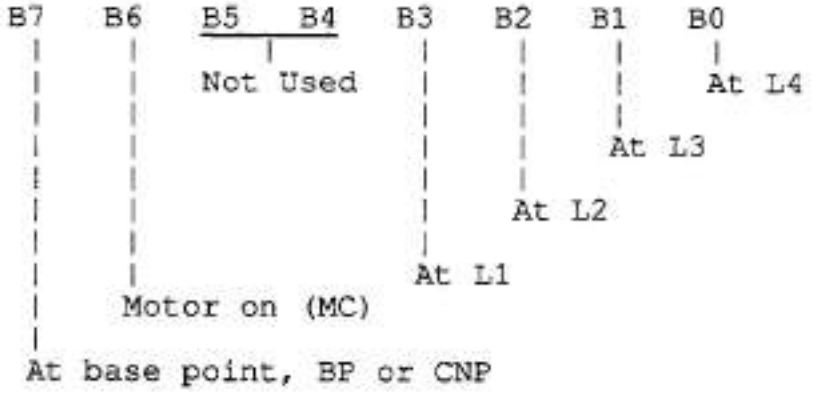
B3	goes to logic "1" after L3 or L4 limit switch inputs have forced a stop. Otherwise it is "0".
B4	goes to logic "1" after L1 or L2 limit switch inputs have forced a stop. Otherwise it is "0".
B5	Is the motor ON signal (inverse of MC input). If the motor loses power, B5 is "1" otherwise it is "0" assuming MC is connected to suitable power sensing circuitry. If MC is left open circuit, then B5 will always be "0". The PPMC-101C will not execute any commands if MC is at logic "0".
B6	Is logic "1" if a motion command was aborted by an emergency stop or decelerating stop command, otherwise it is "0".
B7	Corresponds to the interrupt flag. Whether interrupts are enabled or not, whether a command was aborted or not, B7 is set to "1" at the completion of the command. It is cleared by a second read of the finish status. In multiple PPMC-101C systems sharing a common interrupt, the finish flags of each controller can be polled to see which one generated the interrupt and the interrupt service routine can clear the flag ready for the next command after servicing the interrupt. This is the arrangement on the MSTEP-5 - see the MSTEP.ASM listing & MODE 11.

If all the bits B3-B6 are zero, it indicates that the last command executed to completion and the step count input was actually moved. You can implement various recovery routines on conditions that might arise e.g.:

```
xxx10 IF (D%(2) AND &H08)=&H08 THEN GOSUB aaaa
                               'recover from stop at inner h.s.  limit
xxx20 IF (D%(2) AND &H10)=&H10 THEN GOSUB bbbb
                               'recover from stop at outer limit
xxx30 IF (D%(2) AND &H20)=&H20 THEN GOSUB cccc
                               'notify user there is no motor power
xxx40 IF (D%(2) AND &H40)=&H40 THEN GOSUB dddd
                               'recover from an emergency or decelerating stop
xxx50                          'continue program . . .
```

Obviously this can get quite complicated, but at least all the information you need to make a recovery including the remaining step count (STP#) is returned by the MODE 8 status read.

D%(3) contains the **INPUT STATUS** consisting of a byte of data:



All these bits, with the exception of B4 & B5 which are indeterminate, will be high unless the motor is at the limit switch position. Note that this is an instantaneous read of the limit switch states, if you have overshot a limit, it will not store that information, the input status simply tells you the current state of the limit switch inputs.

The main use of the input status is in determining whether the motor is at a limit, what limit it is and hence performing recoveries or re-calibrations (homing to a reference) through appropriate software routines. The individual limit switch inputs can be separated out with ANDing operations as in the previous example.

D%(4) contains the **OUTPUT STATUS** consisting of a byte of data:

```
B7   B6   B5   B4   B3   B2   B1   B0
 |    |    |    |    |    |    |    |
S1   S2   S3   S4   S5    |    |    Not used
                          |    |
                          |   Direction
                          |
                       Hold signal
```

The output status provides information on the PPMC-101C output state. Bits B7 thru B3 simply reflect the current state of the phase drive outputs S1-S5. B2 is the "HOLD" signal corresponding to HCK on the connector. This goes high 3 milliseconds after the motor comes to a standstill but @B[only if] you have selected switching at standstill on in initialization MODE 12 (D%(8)=1 in initialization). The intent of the external hold output is most likely to let you turn on some sort of electromechanical holding brake as the holding torque with switching on is fairly low.

Bit B1 reflects the last rotation direction, "0" = clockwise, "1"= counterclockwise. Bit B0 is not used and is indeterminate.

STP# contains the remaining step count after a command has aborted. If the command completed STP# will be zero. The sign of STP# is always positive regardless of whether you were going clockwise or counter-clockwise at the time i.e. it is a true remaining step count.

Between FLAG%, D%(2) - D%(4) and STP#, the MODE 8 read status command provides a wealth of information. Note that the byte formats of D%(2)- D%(4) are identical to those described in the PPMC-101C data sheet (Appendix A) under finish status, input status and output status respectively. You may find the data sheet useful in further understanding the various status conditions. MODE 8 actually does 4 consecutive read register operations and returns every conceivable status condition in one operation, this is simply a convenience, you can use what you want and ignore the rest.

Entry Data:

MD% = 8	Read status
D%(0)	Selects Channel A (0) or Channel B (1)
D%(1) thru (9)	Value irrelevant
STP#	Value irrelevant
FLAG%	Value irrelevant

Exit Data:

D%(0) thru (1)	Unchanged
D%(2)	Finish status
D%(3)	Input status
D%(4)	Output status
D%(5) thru (9)	Unchanged
STP#	Remaining pulses (0 - 16,777,215)
FLAG%	0 if executed OK 1 if motor busy 2 if driver not initialized on Channel A 3 if driver not initialized on Channel B 4 if MD% <0 or >12 5 if hardware error 10 if D%(0) not 0 or 1

Note that if FLAG%=1 then D%(2) thru D%(4) and STP# will be returned unchanged as the PPMC-101C is unable to execute a status command unless the motor is at standstill.

A typical program entry preceding MODE 8 might read as follows:

```
xxx10 MD% = 8                                          'select MODE number
xxx20 D%(0) = 0                                        'select Channel A
xxx30 CALL MSTEP (MD%, D%(0), STP#, FLAG%)
                                                       'read status
xxx40 IF FLAG%=1 THEN GOTO
xxx30                                                  'keep trying until motor stops
xxx50 IF FLAG%<>0 THEN PRINT "Error in MODE 8, # ";FLAG%:STOP
xxx60                                                  'continue program, process
                                                            D%(2)-D%(4), STP#
                                                            etc.
```

NOTE: The loop in line xxx40 may not always be action you might want on finding the motor busy. Change it to suit.

4.10 MODE 9: READ ENCODER

MODE 9 does a simple read of the 24-bit up/down encoder counter (either Channel A or B specified by D%(0)), and returns the data in STP#. The 24-bit counter data is treated as signed 2's complement and returned to STP# as a number in the range -8,388,608 to +8,388,607, positive corresponding to clockwise and negative to counter-clockwise. A 1000 line/rev. encoder, such as Metrabyte's STEP-ENC1, will thus travel about 8,388 revolutions with a resolution of about a third of a degree without overrunning the counter capacity. For more information on encoders, see Section 4.

Note that the counter is not latched, it is read directly, and since the three counter bytes are read sequentially, you will most likely return erroneous data if you read the encoder while it is moving. It is essential to read it only at standstill, but see the programming trick below.

Resetting of the encoder counter is performed by initialization MODE 12. The assumption here is that you will not need to reset the encoder after configuring the system, so in the interests of simplicity, the driver has omitted this capability except through MODE 12. If you do need to clear the encoder counters without re-initializing the stepper motor controller, it is very simple to do with a few OUT's, see the example in Section 4.3

Entry Data:

MD% = 9	Read encoder
D%(0)	Selects channel A (0) or channel B (1)
D%(1) thru (9)	Value irrelevant
STP#	Value irrelevant
FLAG%	Value irrelevant

Exit Data:

D%(0) thru (9)	Unchanged
STP#	Encoder data (-8,388,608 to +8,388,607)
FLAG%	0 if executed OK 2 if driver not initialized on Channel A 3 if driver not initialized on Channel B 4 if MD% <0 or >12 10 if D%(0) not 0 or 1

A typical program entry preceding MODE 9 might read as follows:

```
xxx10 MD% = 9                                          'select MODE number
xxx20 D%(0) = 0                                        'select Channel A
xxx30 CALL MSTEP (MD%, D%(0),STP#, FLAG%)
                                                       'read encoder
xxx40 IF FLAG%<>0 THEN PRINT "Error in MODE 9, # ";FLAG%:STOP
xxx50                                                  'continue program, readSTP#
                                                            etc.
```

NOTE: Since this read of the encoder is totally independent of the stepper controller hardware, you may not know if the encoder is moving. Obviously if the encoder is mechanically coupled to the stepper motor, you could use read status MODE 8 to find out if the stepper motor had stopped and know with certainty that it is safe to read the associated encoder. If the encoder is not connected to the stepper motor e.g. used as an operator reference input, then reading it two or more times in a row is a good way to find out if it is at rest. If the 2 values returned are equal, you can assume that the encoder is at rest or moving so slowly that the data is valid anyway. Try something like this:

```
xxx10 MD% = 9                                          'select MODE number
xxx20 D%(0) = 1                                        'select Channel B
xxx30 RDG1# = 0                                        'declare reading 1 variable
xxx40 RDG2# = 0                                        'and reading 2 variable
xxx50 CALL MSTEP (MD%, D%(0), RDG1#, FLAG%)
                                                       'read encoder 1st time
```

```
xxx60 CALL MSTEP (MD%, D%(0), RDG2#, FLAG%)
                                          'read encoder 2nd.  time
xxx70 IF RDG1# <> RDG2# THEN GOTOxxx50
                                          'keep trying till it stops
                                               moving
xxx80                                     'Continue, process data from
                                               RDG1# . . .
```

If you end up doing this maneuver a lot, you might want to consider modifying the driver and have it do it for you instead e.g return FLAG%=1 if the encoder is moving.

4.11 MODE 10: LOAD DIVIDER

MODE 10 sets the division ratio of the external stepper clock divider. It applies only if you have selected operation from the external 100 KHz on-board clock, D%(7)=1 in initializing MODE 12, otherwise it will do nothing for you (see MODE 12 for the effective step rate). The actual divider ratio will be one greater than this parameter. Thus for a D%(1) range of 1-255, the divider ratio will be 2-256.

Entry Data:

MD% = 10	Load external divider
D%(0)	Selects Channel A (0) or Channel B (1)
D%(1)	Sets divider ratio
D%(2) thru (9)	Value irrelevant
STP#	Value irrelevant
FLAG%	Value irrelevant

Exit Data:

D%(0) thru (9)	Unchanged
STP#	Unchanged
FLAG%	0 if executed OK 2 if driver not initialized on Channel A 3 if driver not initialized on Channel B 4 if MD% <0 or >12 10 if D%(0) not 0 or 1 11 if D%(1) <1 or >255

A typical program entry preceding MODE 10 might read as follows:

```
xxx10 MD% = 10                            'select MODE number
xxx20 D%(0) = 0                           'select Channel A
xxx30 D%(1) = 100                         'divider ratio = 100
xxx40 CALL MSTEP (MD%, D%(0),STP#, FLAG%)
                                          'load divider
xxx50 IF FLAG%<>0 THEN PRINT "Error in MODE 10, # ";FLAG%:STOP
```

4.12 MODE 11: ENABLE/DISABLE INTERRUPT

Either or both PPMC stepper motor controllers are capable of generating an interrupt to signify completion of certain motion control commands. BASIC cannot be used to program interrupt service routines as there is no ON INTERRUPT construct. If you wish to make use of the MSTEP-5 interrupt capabilities, you will have to resort to some assembly language programming and modification of the driver. MODE 11 does a lot of the groundwork for you in installing an example routine and enabling/disabling it. While it is impossible to predict your requirements in terms of what you might want the interrupt service routine to do, as an example, this MODE installs a sample interrupt service routine (label INTH: in the MSTEP.ASM source) which simply sounds a beep on generation of an interrupt. The MSTEP.ASM source can be modified to change the INTH: service routine to your needs and re-assembled.

MODE 11 performs the following:

1. Disables interrupts on the selected level (2-7).
2. Installs interrupt vectors to INTH: for that level.
3. Either enables or disables hardware interrupts on the selected level according to D%(2).

WARNING It is your responsibility to avoid conflict with other peripherals when using interrupts - see below.

It is important to note that MODE 11 does not save and restore vectors to any previous interrupt service routine that may have been serviced on the selected level. If you need this capability for sharing devices on the same level, you need to expand and modify the driver code. The assumption has been made that if you are going to use interrupts at all, you will dedicate one of the hardware interrupt levels 2 thru 7 to the MSTEP-5. If interrupts are disabled (default power up & initializing conditions) then the MSTEP-5 interrupt output is tri-stated and will not interfere with any device on the level. The operating level is selected by the jumper block labelled "IRQ LEVEL" on the MSTEP-5 board.

If interrupts are enabled while the selected motor is busy, they will not take effect until execution of the following command. Only certain motion control commands are able to generate an interrupt on completion. These are:

MODE 2: Single step or jog.
MODE 3: Accelerate/decelerate forSTP# pulses.
MODE 4: Constant speed forSTP# pulses.
MODE 5: Constant speed to outer limit switch.
MODE 6: High speed to inner limit switch.
MODE 7: Move to base or reference point.

Because the interrupts from Channels A and B are OR'd together by the MSTEP-5 hardware, if both are enabled you will need to poll the status of both controllers to determine which one generated the interrupt. This requires a read finish status register command which in turn will clear the interrupt signal from the requesting controller. This may be part of your Interrupt Service Routine (as in the example handler) or provided by your foreground program by a call to MODE 8 as dictated by your needs.

The following list provides the normal IBM reserved functions for Hardware Interrupts 2 thru 7.

LEVEL	FUNCTION
2	Usually free PC & PC/XT, cascade input on PC/AT
3	COM2: (if installed)
4	COM1: (if installed)
5	LPT2: (if installed)
6	Floppy disk (always used)
7	LPT1: (if installed)

If you do not have the particular hardware item installed, it is safe to use that level. Generally, Levels 2 or 5 work out well on PCs and PC/XTs, and Level 5 on PC/ATs. Never use Level 6 as you will mess up the floppy disk operation. Levels 3, 4, and 7 may be available depending on what equipment you have installed and whether you are using it you can often share with a peripheral that your program will not be using as long as the peripheral's interrupt hardware is disabled.

Entry Data:

MD% = 11	Enables/disables interrupt
D%(0)	Selects Channel A (0) or Channel B (1)
D%(1)	Value irrelevant
D%(2)	Enables (1) or disables (0) interrupt
D%(3)	Interrupt level (2 - 7); if D%(2) = 0 (disable), value of D%(3) is irrelevant.
D%(4) thru (9)	Value irrelevant
STP#	Value irrelevant
FLAG%	Value irrelevant

Exit Data:

D%(0) thru (9)	Unchanged
STP#	Unchanged
FLAG%	0 if executed OK 2 if driver not initialized on Channel A 3 if driver not initialized on Channel B 4 if MD% <0 or >12 10 if D%(0) not 0 or 1 12 if D%(2) not 0 or 1 13 if D%(2)=1 and D%(3) <2 or >7

A typical program entry preceding MODE 11 might read as follows:

```
xxx10 MD% = 11                                  'select MODE number
xxx20 D%(0) = 0                                 'select Channel A
xxx30 D%(2) = 1                                 'enable interrupt
xxx40 D%(3) = 5                                 'on level 5
xxx50 CALL MSTEP (MD%, D%(0),STP#, FLAG%)
                                                'enable interrupt
xxx60 IF FLAG%<>0 THEN PRINT "Error in MODE 11, # ";FLAG%:STOP
```

```
xxx70                                              'continue program . . .

yyy10  . . .  and if you want to disable interrupts at any point
yyy20 MD% = 11                                     'select MODE number
yyy30 D%(0) = 0                                    'select Channel A
yyy40 D%(2) = 0                                    'disable interrupt
yyy50 CALL MSTEP (MD%, D%(0),STP#, FLAG%)
                                                   'disable interrupt
yyy60 IF FLAG%<>0 THEN PRINT "Error in MODE 11, # ";FLAG%:STOP
yyy70                                              'continue program . . .
```

4.13 MODE 12: INITIALIZATION

MODE 12 initializes the driver and MSTEP-5 hardware and must be executed before using any of the other MODEs. Initializing must be done separately for each Channel A and B and performs the following operations:

1. Disables hardware interrupts.
2. Checks and stores the MSTEP-5's base I/O address.
3. Checks for presence of MSTEP-5 board at this address.
4. Checks the validity (range) of all initialization data.
5. Resets the PPMC-101C controller.
6. Loads the controller with initialization data.
7. Clears the encoder counter.
8. Sets initialization flags to enable selection of other MODEs.

The base I/O address is checked to be in the legal range of 256 1016 (Hex 100 - 3F8) for the IBM P.C. If not an error, Exit #19 will occur. If OK, the base I/O address is stored for use by other MODEs on re-entry to the CALL.

A short read/write test is made to the MSTEP-5 control register which is sufficient to detect the presence or absence of the board at the specified I/O address. If no board is detected (absent board, wrong base I/O address), Error #5, hardware error is returned. Error #5 may also be returned by this MODE or any other MODE if the PPMC-101C controller fails to perform correct handshakes on inputs and outputs of commands and data (see Appendix A). In this case, Error #5 may be indicative of a failure of the PPMC-101C controller especially if one channel works and the other does not.

All the initialization data is checked to be within valid limits. Any of the initializing data variables D%(N) that is outside acceptable limits will return error code 10+N (this is true for all MODEs). Also in the interests of consistency D%(0) always specifies the channel selected (0=A, 1=B) and D%(1) the constant speed rate in all MODEs. Initialization MODE 12 makes use of all 10 elements of D%(*) and is the only MODE to require so much input data.

If initialization is successful, any other MODE may be selected on subsequent CALL's. **Trying to select any other MODE before performing initializing MODE 0 will give rise to error # 2 or 3 depending on whether Channel A or B is uninitialized.**

Entry Data:

MD% = 12	(Value Irrelevant)
D%(0)	Channel selector, A = 0, B = 1
D%(1)	Start up rate divider, RA, range 20-255
D%(2)	High speed run rate divider, range 20-255
D%(3)	Acceleration/deceleration pulses, 4-8,160
D%(4)	Motor type, valid codes 1,2 or 3 (see below)
D%(5)	Full step (0) or half step (1) MODE (see below)
D%(6)	Logic polarity of S1-5, 1 = positive, 0 = inverted
D%(7)	Clock source, 0 = internal, 1 = on board external, 2 = external user input
D%(8)	Power switching at standstill, 0 = off, 1 = on
D%(9)	Base I/O address, valid range 100-3F8 hex
STP#	Value irrelevant
FLAG%	Value irrelevant

Exit Data:

D%(0) thru (9)	Unchanged
STP#	Unchanged
FLAG%	0 if initialization OK 4 if MD% <0 or >12 5 if hardware error, no board, incorrect I/O address 10+N if D%(N) not in valid range

A typical start of program initializing sequence would be as follows:

```
100 DEF SEG = &H6000                              'segment to load driver
110 BLOAD "MSTEP.BIN",0                           'load it at zero offset
120 MSTEP = 0                                     'declare variables
130 MD% = 12                                      'select initialization
140 DIM (9)                                       'declare data array
150STP# = 0                                  'step count variable
                                                        (double-precision)
160 FLAG% = 0                                     'declare error variable
170 D%(0) = 0                                     'select Channel A
180 D%(1) = 255                                   'start up rate, 49 pps
190 D%(2) = 80                                    'run rate, 156 pps
200 D%(3) = 1000                                  'acceleration/deceleration
                                                        pulse count
210 D%(4) = 2                                     'motor code 4 phase
220 D%(5) = 0                                     'full step
230 D%(6) = 0                                     'inverted S1-5 outputs
240 D%(7) = 0                                     'internal clock
250 D%(8) = 0                                     'switching off at standstill
260 (9) = &H300                                   'base I/O address (hex 300)
270 CALL MSTEP (MD%, D%(0),STP#, FLAG%)
                                                  'initialize
280 IF FLAG%<>0 THEN PRINT "Error in initializing # ";FLAG%:STOP
290                                               'continue program
```

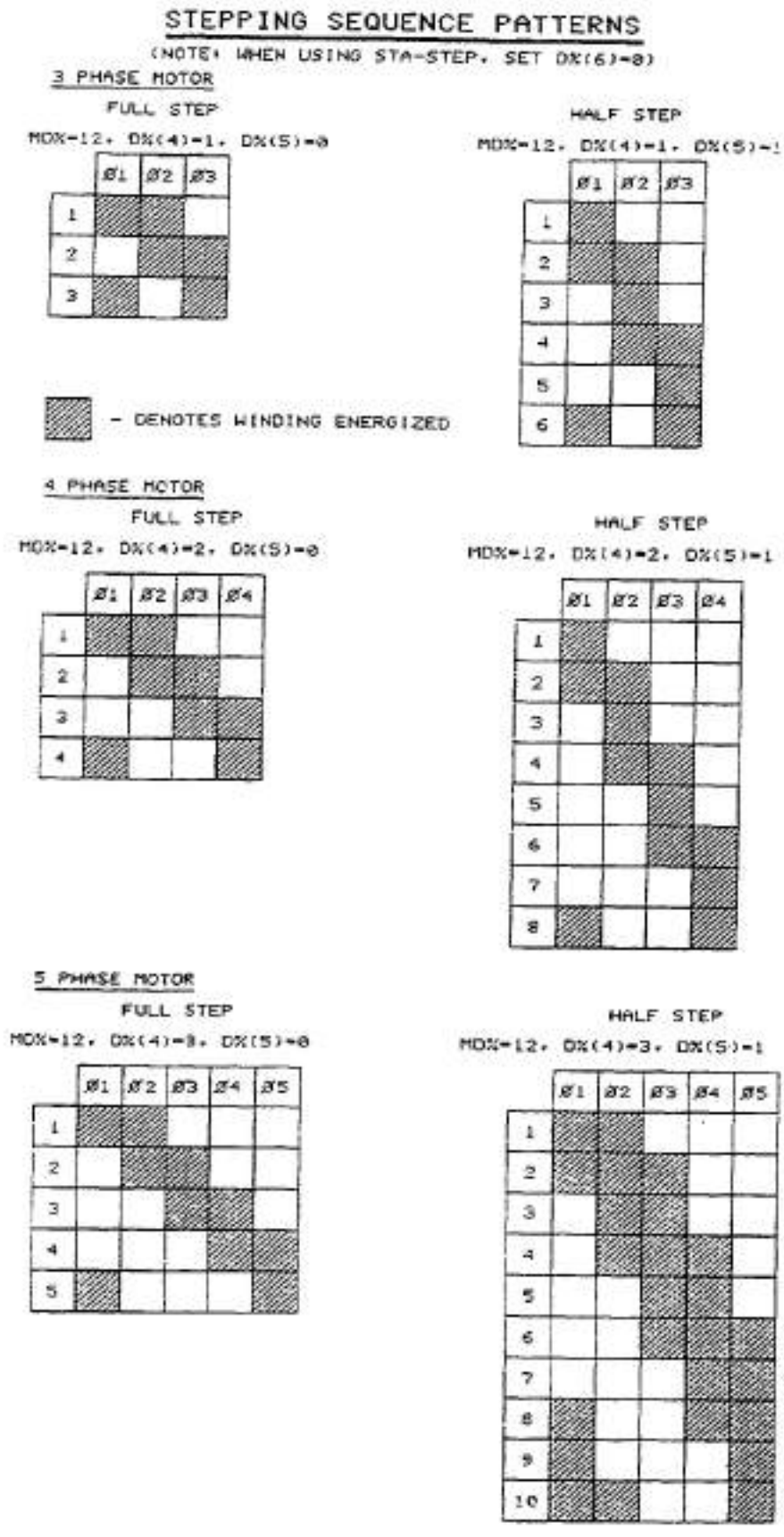


Figure 4-1. Stepping sequences for 3-, 4-, and 5-phase full/half step.

There is quite a lot of data to be provided at initialization, but it is only required once! Let's look at the effect of the initializing parameters in more detail:

D%(0) Controls which channel is selected. If D%(0) = 0 then the CALL operates on Channel A motor and encoder, if D%(0) = 1 then the CALL operates on channel B motor and encoder. Note that each each channel must be initialized individually and may have different initializing parameters, motor types etc.

D%(1) Controls the start up rate of the motor in MODEs 3 and 6 which involve acceleration and deceleration. (D%(1) corresponds to RA in the PPMC data sheet). The start up rate with internal clock (D%(7)=0):

Rate = 12,500 / D%(1) steps per sec.

With external user clock input (D%(7)=2):

Rate = Ext. freq. / D%(1) steps per sec.

With onboard 100KHz external clock (D%(7)=1):

Rate = 100,000 / (D%(1)*(X+1)) steps per sec.

where X = external divider ratio set in MODE 10.

The start up rate should be chosen slow enough so that the motor does not skip pulses on getting moving. If in doubt, set D%(1) = 255 (slowest).

D%(2) Controls the high speed run rate of the motor in MODEs 3 and 6 which involve acceleration and deceleration. For internal clock (D%(7)=0):

Rate = 12,500 / D%(2) steps per sec.

With external user input clock (D%(7)=2):

Rate = Ext. freq. / D%(2) steps per sec.

With onboard 100KHz external clock (D%(7)=1):

Rate = 100,000 / (D%(2)*(X+1)) steps per sec.

where X = external divider ratio set in MODE 10 The high speed run rate should be chosen so that the motor does not skip pulses at high speed. This usually requires some judicious experimentation. If you start off with D%(2)=150 and then reduce it until the motor shows signs of distress and then back off a a little, it will generally work out OK.

D%(3) Controls the number of steps that the motor will accelerate from the start rate to the high speed run rate and vice versa decelerate. The trapezoidal velocity profile and the controlling parameters are shown below in Fig.3.4. The acceleration/deceleration step count D%(3) may be anywhere from 4 to 8,160.

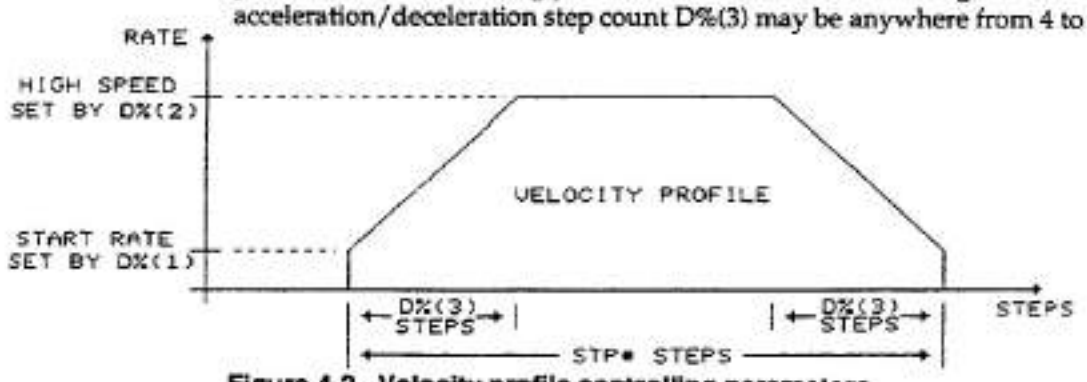


Figure 4-2. Velocity profile controlling parameters.

D%(4) Sets the motor code (1 - 3) depending on the winding arrangement. 3-phase motors are Code 1, 4-phase motors are Code 2, and 5-phase motors are Code 3. Another way to determine the motor code is to determine the required switching sequence as shown in Figure 4-1. Note that 3-phase motors use only phase outputs S1-S3, 4-phase use S1-S4 and 5-phase use all S1-S5. The motors additionally may be operated in full-step or half-step as set by D%(5). If you are not using the PPMC-101C phase outputs (e.g. in conjunction with an STA-STEP) then the motor code is irrelevant. D%(4) corresponds to the motor code parameter on the PPMC-101C data sheet.

D%(5) Selects the stepping MODE, full-step (0) or half-step (1). It corresponds to the excitation parameter of the PPMC-101C data sheet (see Appendix A). A standard 200 step/revolution motor (e.g. STEP-MOT1) will step 1.8 degree increments in full-step MODE or 0.9 degree increments (400 steps/revolution) in half-step. The half-step MODE offers finer resolution and smoother stepping but at the cost of reduced torque and maximum operating speed compared to full-step.

D%(6) Controls the logic polarity of the S1-S5 phase drive outputs. D%(6)=1 selects positive true logic and D%(6)=0 selects active low negative logic. The choice here depends on the driving hardware. For the STA-STEP set D%(6)=0. If you are not using the phase outputs S1-S5, the choice is irrelevant.

D%(7) Selects the clock source: D%(7) = 0 : Selects internal PPMC 12.5KHz source = 1 : Selects on board 100KHz + divider = 2 : Selects external user input For most purposes the internal clock source will be adequate. If you need step rates above 625 pps or synchronized stepping of both axes, use one of the external options.

D%(8) Controls switching at standstill, D%(8) = 0 turns switching off, D%(8) = 1 turns it on. At standstill the motor will normally have maximum current flowing in the windings which causes the greatest ohmic heating and temperature rise of the motor. One way of reducing this heating is to chop the phase drive outputs when the motor has reached standstill. With switching on, chopping commences 100 milliseconds after the motor reaches standstill at a frequency of about 2.2KHz and a duty cycle of 30%. As soon as the motor is commanded to move, chopping ceases and will automatically resume on standstill. Chopping tends to reduce the holding torque, raise the driver transistor switching losses, reduce heating of the motor and may produce audio noise from the motor. There are thus advantages and disadvantages associated with this feature.

D%(9) Selects the base I/O address which must correspond to the dipswitch setting on the board. IBM PCs and XTs decode addresses from hex 200 - 3FF. IBM ATs decode addresses from hex 100 - 3FF. The driver checks for a valid address in the range 100-3F8 hex although the hardware can physically be set from address 0 - 3F8 hex.

In our examples of programming all the different MODEs, we have consistently used D%(*) as the data array. You may prefer to keep your initialization parameters in another array e.g. IX%(*) and not overwrite its values when subsequently selecting other MODEs. This is easy to implement:

```
xxx00 DIM IX%(9), (9)
xxx10 MD% = 12
xxx20 IX%(0) = 1 : IX%(1) = 200 : IX%(2) = 50 etc.
```

then:

```
xx100 CALL MSTEP (MD%, IX%(0),STP#, FLAG%)    initialize
```

after this use D%(*) for other MODEs:

```
xx200 MD% = 3
xx210 D%(0) = 0
xx220STP# = 999
xx230 CALL MSTEP (MD%, D%(0),STP#, FLAG%)
                                              'accel/decel
. . . etc.
```

4.14 PROGRAMMER'S SHORT SUMMARY

PROGRAMMER'S QUICK REFERENCE

CALL MSTEP (MDZ, DZ(0), STP#, FLAGZ)

MODE MDZ	FUNCTION	DATA ARRAY DZ(-) DZ(0)	DZ(1)	DZ(2)	DZ(3)	DZ(4)	DZ(5)	DZ(6)	DZ(7)	DZ(8)	DZ(9)	STP# STEPS
0	EMERGENCY STOP	SELECTS CHANNEL A=0, B=1	—	—	—	—	—	—	—	—	—	—
1	DECELERATING STOP		—	—	—	—	—	—	—	—	—	—
2	JOG SINGLE STEP		—	—	—	—	—	—	—	—	—	SIGN SETS CW/CCW
3	ACCEL/DECEL STP# STEPS		—	—	—	—	—	—	—	—	—	STEP COUNT CW/CCW
4	CONSTANT SPEED STP# STEPS		RATE 28-255	—	—	—	—	—	—	—	—	STEP COUNT CW/CCW
5	CONSTANT SPEED TO OUTER LIMIT		RATE 28-255	—	—	—	—	—	—	—	—	SIGN SETS CW/CCW
6	ACCEL/DECEL TO INNER LIMIT		—	—	—	—	—	—	—	—	—	SIGN SETS CW/CCW
7	CONSTANT SPEED TO HOME POINT		RATE 28-255	—	—	—	—	—	—	—	—	SIGN SETS CW/CCW
8	READ STATUS		—	RETURN FINISH STATUS	RETURN INPUT STATUS	RETURN OUTPUT STATUS	—	—	—	—	—	RETURNS STATUS
9	READ ENCODER		—	—	—	—	—	—	—	—	—	RETURNS ENCODER DATA
10	LOAD EXTERNAL CLK DIVIDER		DIVIDER DATA 1-255	—	—	—	—	—	—	—	—	—
11	ENABLE OR DISABLE INTERRUPT		—	ENABLE/DISABLE E-I	LEVEL 2 - F	—	—	—	—	—	—	—
12	INITIALIZE		START RATE 28-255	RUN RATE 28-255	ACCEL/DECEL STEPS 1-65535	MOTOR CODE 1 - 3	FULL / HALF STEP 0-F, 1-H	LOGIC LEVEL 0-INV, 1-TRUE	CLOCK SOURCE 0-INT, 1-EXT	INTERRUPT AT STANDSTILL 0-OFF, 1-ON	BASE ADDR. 100H-3F0H	—

ERROR FLAGZ:
- 0 - NO ERROR, O.K.
- 1 - MOTOR BUSY
- 2 - DRIVER NOT INITIALIZED ON CHANNEL A
- 3 - DRIVER NOT INITIALIZED ON CHANNEL B
- 4 - MODE NUMBER <0 OR >12
- 5 - HARDWARE ERROR
- 6 - STEP COUNT OUT OF RANGE <-16,777,215 OR >16,777,215
- 7 - MOTOR ALREADY AT STANDSTILL
- 10+N - ERROR IN VALUE OF DZ(N)

4.15 INTERPRETED BASIC (GW, COMPAQ, IBM, ETC.)

Example

BASIC Call: CALL mstep(MD%, D, STP#%, FLAG%)

BASIC Declaration: NONE NECESSARY IN BASIC SOURCE CODE. However, a "BLOAD" (Binary load of .BIN file) of the binary file containing the external subroutine must be done prior to calling that subroutine.

Example Program Illustrating a BASIC CALL:

```
10 '************************************************************
20 '*                                                          *
30 '*          MSTEP-5 Demonstration program                   *
40 '*          Keithley MetraByte Corporation                  *
50 '************************************************************
60 '
70 SCREEN 0,0,0:WIDTH 80:KEY OFF:CLS
80 LOCATE 1,1,0                                  'turn off cursor
90 LOCATE 25,1:PRINT"Metrabyte MSTEP-5 Demo program Loading
                                        MSTEP.BIN driver";:LOCATE 1,1
100 '
110 '--- Load MSTEP.BIN driver ----------------------------------------
120 CLEAR, 48000!                                'set workspace to 48000 bytes
                                                      (for example)
130 DEF SEG = 0                                  'set zero prior to PEEK's to
                                                      zero segment
140 LS = 256 * PEEK(&H511) + PEEK(&H510)
                                                 'Basic's data segment
150 SG = LS + 48000!/16                          'end of Basic's data segment
160 DEF SEG = SG                                 'this is where to load
                                                      MSTEP.BIN driver
170 BLOAD "MSTEP.BIN",0 'load driver with zero offset
180 '

          .
          .
          .

330 '--- Declare other CALL variables ---------------------------------
340 MD% = 12                                     'mode number
350 MSTEP = 0                                    'call offset
360 FLAG% = 0                                    'call error flag variable
370 STP#  = 0                                    'step count (must be double
                                                      precision)
380 '
```

```
390 '--- Display menu ------------------------------------------------

          .
          .
          .

2590 '--- MODE 12: Initialize ------------------------------------------
2600 CLS
2610 LOCATE 25,1:PRINT"Metrabyte MSTEP-5 Demo program
                                        Initialize - mode 12";:LOCATE 1,1
2620 GOSUB 3250   'display channel selected

          .
          .
          .

2830 CALL MSTEP (MD%,D%(0),STP#,FLAG%)
2840 IF FLAG%<>0 THEN GOSUB 3300:IF E%=0 THEN GOTO 2600
2850 RETURN

          .
          .
          .

3460 '
3470 END
```

4.16 QUICKBASIC

Example

BASIC Call: CALL QBPIOINT(MD%, VARPTR(D%(0)) STP#%, FLAG%)

BASIC Declaration: DECLARE SUB QBPIOINT (MD%,BYVAL DUMMY%, FLAG%)
The Declaration tells QuickBASIC that the subroutine expects three arguments and that the middle argument is to be passed by value. Remember that BASIC normally passes all arguments by reference (address). This is the only method for passing an array to a subroutine in BASIC: passing the value of the address of the array in effect passes the array by reference. To make use of the callable assembly routine, a ".QLB" (Quick Library) file is created out of the original .ASM source file. Although the format of the subroutine is identical to those used by interpreted BASIC packages, both the Quick BASIC integrated development environment (QB.EXE) and the command line complier (BC.EXE) expect the subroutine to be in a specially formatted .QLB library file. Unlike interpreted BASIC packages, Quick BASIC actually links to the assembly .QLB library file so it is not necessary to include the "jmp QBPIOINT" instruction at location 0 (of the source file) as in interpreted BASIC.

Example Program Illustrating a QuickBASIC CALL:

```
'************************************************************************
'*                                                                      *
'*              MSTEP-5 Demonstration program                           *
'*                     for Quick Basic                                  *
'*              Keithley MetraByte Corporation                          *
'*                                                                      *
'************************************************************************
'
DIM L$(14), P$(9), U$(9), D%(9), U(9)
COMMON SHARED D%()
DECLARE SUB QBMSTEP (MD%, BYVAL DUMMY%, STP&, FLAG%)
SCREEN 0, 0, 0: WIDTH 80: KEY OFF: CLS
LOCATE 1, 1, 0                          'turn off cursor
LOCATE 25, 1: PRINT "Metrabyte MSTEP-5 Demo program Loading MSTEP.BIN
                    driver";
LOCATE 1, 1
'
'--- Declare arrays ---------------------------------------------------
'
'Default parameters for initializing controllers
P$(1) = "              Start rate divider: ": U(1) = 255  'slowest
P$(2) = "     High speed run rate divider: ": U(2) = 100  'medium
P$(3) = " Acceleration/deceleration steps: ": U(3) = 200  '200 steps
P$(4) = "                      Motor type: ": U(4) = 2    '4 phase
P$(5) = "                      Excitation: ": U(5) = 0    'full step
P$(6) = "                   Logic polarity: ": U(6) = 0    'inverted
P$(7) = "                    Clock source: ": U(7) = 0    'internal
P$(8) = "         Switching at standstill: ": U(8) = 0    'off
P$(9) = "                Base I/O address: ": U(9) = &H340
'
'--- Declare other CALL variables -------------------------------------
MD% = 12                        'mode number
D%(0) = 0                       'set channel A as default
FLAG% = 0                       'call error flag variable
STP& = 0                        'step count (must be double precision)
'--- Display menu -----------------------------------------------------

     .
     .
     .

'--- MODE 12: Initialize ----------------------------------------------
CLS
LOCATE 25, 1: PRINT "Metrabyte MSTEP-5 Demo program
                             Initialize - mode 12"; : LOCATE 1, 1
     .
     .
     .
```

```
CALL QBMSTEP(MD%, VARPTR(D%(0)), STP&, FLAG%)
IF FLAG% <> 0 THEN GOSUB 3250: IF E% = 0 THEN GOTO 2550
RETURN

      .
      .
      .

'
END
```

4.17 MULTIPLE MSTEP-5s IN ONE SYSTEM

What if you wish to operate more than one MSTEP-5 in a system? To avoid conflicts, each MSTEP-5 must have a different base I/O address. For simultaneous operation using interrupts select a different interrupt level for each board. It is possible to share an interrupt level if the operations can be sequential instead of simultaneous. Each board must also be assigned its own CALL routine. To do this start by loading the MSTEP.BIN routine at different locations in memory:

```
xxx10 SG1 = &H3000
xxx20 SG2 = &H4000
xxx30 DEF SEG = SG1
xxx40 BLOAD "MSTEP.BIN",0
xxx50 DEF SEG = SG2
xxx60 BLOAD "MSTEP.BIN",0
```

Now the CALL appropriate to each board can be entered as required. Note that each CALL is preceded by a DEF SEG appropriate to that board:

```
yyy10 DEF SEG = SG1
yyy20 CALL MSTEP (MD1%, D1%(0), STP1#, FLAG1%)
yyy30 DEF SEG = SG2
yyy40 CALL MSTEP (MD2%, D2%(0), STP2#, FLAG2%)
    'etc.
```

* * * * *

CHAPTER 5

ENCODER CHANNELS

5.1 OPTICAL INCREMENTAL SHAFT ENCODER

Shaft encoders fall into two categories, incremental and absolute. Absolute encoders use a Gray coded disc and produce a specific code for the shaft position using multiple tracks and light sources/detectors on the disc. Incremental encoders can only measure changes in position but are simpler in construction and generally capable of finer resolution. This is the type that is used with the MSTEP5.

An incremental encoder consists of a disc that has radial transparent and opaque sectors with 2 or 3 light sources and detectors. The disc may typically have anywhere from 90 to 1200 radial lines, giving a corresponding number of output pulses per revolution. By using two detectors that are slightly displaced from each other, two pulse trains are output that are 90 degrees displaced in phase from each other. Encoders with this type of output are called quadrature encoders, the typical output signal being shown below:

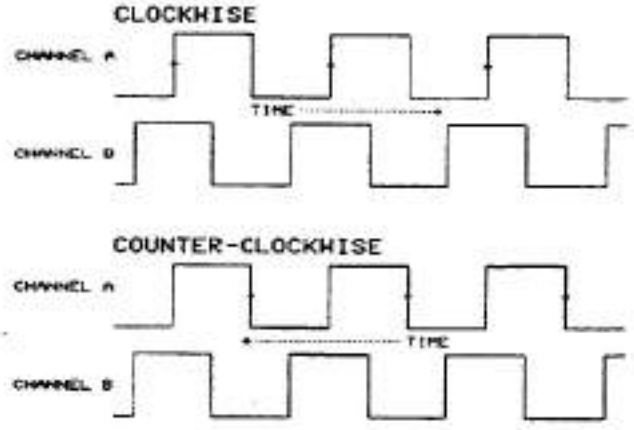


Figure 5-1. Quadrature encoder output signals.

Note that when the encoder is rotated in a clockwise direction, the positive transitions (0-1) of channel A occur while the channel B signal is high, if the encoder is rotated counter-clockwise, the positive transitions of channel A occur when the channel B signal is low. If the channel B signal was connected to the up/down control input of an up/down counter, and the channel A signal connected to the counter clock, the cumulative count would always reflect the displacement of the encoder from its initial position.

It is common for an incremental encoder to include a third output which produces a pulse once every revolution. This is known as an index pulse. The STEP-ENC1 has an index pulse output, although the MSTEP5 board makes no use of it. Usually the low level outputs from the photodetectors are signal conditioned in the encoder to produce TTL or CMOS compatible output signals, also complimentary signals may be provided to help in minimizing the effects of external noise. Some encoder interface circuits (such as MetraByte's M5312) count all edges of the waveforms and so get 4 times the resolution of a simple circuit that only counts 1 edge e.g. the positive edge. The MSTEP5 circuitry is

of the latter type and makes up for the lower resolution with improved noise immunity and simpler hardware.

In addition to reduced resolution, this simpler approach is somewhat more susceptible to false readings generated when the direction is changed. This can be significant when the encoder is used to measure the movement of a stepper motor. Since many stepper-motor systems produce an under-damped overshoot when coming to a stop, several false steps due to several directional changes are possible. For this reason, it is important that proper damping be introduced into the system to reduce or prevent these under-damped overshoots. See the references of Paragraph 7.6 for additional information about damping.

Should damping prove impractical, use of a MetraByte M5312 Board to read the encoder is recommended.

Optical incremental shaft encoders are fairly robust and are capable of detecting very small displacements. The STEP-ENC1 has a 1000 pulse/revolution output and so will produce an output pulse about every third of a degree of rotation. The main enemies of optical encoders are mechanical shock, electrical noise and low temperatures which may cause ice crystals to form on the encoder disc and interfere with correct operation.

5.2 MSTEP5 ENCODER INTERFACE

One of the commonest problems with encoders is electrical noise. If the encoder is directly connected to a high speed up/down counter, small noise spikes or noisy edges might induce false counts. A simple, but effective method of eliminating this problem is to pass the encoder outputs through a digital noise filter. The waveform below shows what a "real world" encoder signal might look like. Short spikes and transients on the waveform are to be expected from motor switching currents etc.

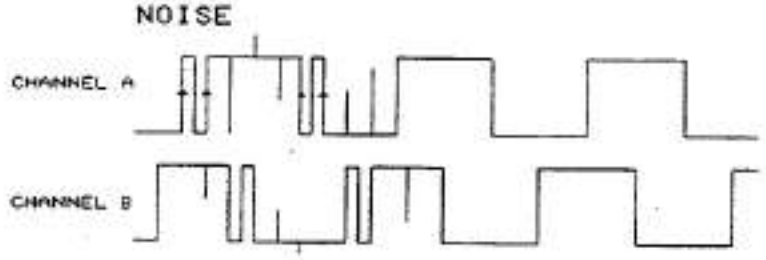


Figure 5-2. Effects of noise on encoder output.

The MSTEP5 uses a sampling digital filter on each channel of the two encoder inputs. The filter is implemented in a PAL device (programmable array logic) for each encoder and the equivalent circuit for each encoder input is shown below:

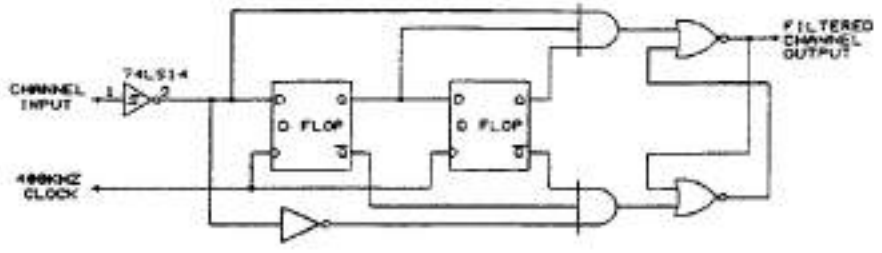


Figure 5-3. Encoder digital filter.

An internal 400KHz clock is used to sample the signal. The two D flops form a 2 stage shift register. When all outputs of the register and the input signal are at the same state, one of the AND gates is enabled and flips the output latch. The result is that the output signal follows the input signal, but is 2 clock pulses (5 microseconds) behind. Short transients, less than 2 clock periods or 5 microseconds won't make it through the shift register. The price paid for this filter is that it cannot handle pulse rates from the encoder much faster than 1/16 of the clock period - in this case 25KHz. This corresponds to a rotational speed for a 1000 line/rev. encoder such as the STEP-ENC1, of 1500 r.p.m. which at least as far as stepper motor speeds are concerned is not particularly restrictive.

The filtered signals are passed to 3 cascaded 8 bit up/down counters. The 24 bit data is converted by the MSTEP.BIN driver to a corresponding 23 bits + sign double precision real. This means that you have a range of -8,388,608 to + 8,388,608 counts or more simply about +/-8000 revolutions from the initial position with a resolution of about one third of a degree using the 1000 pulse/rev. STEP-ENC1.

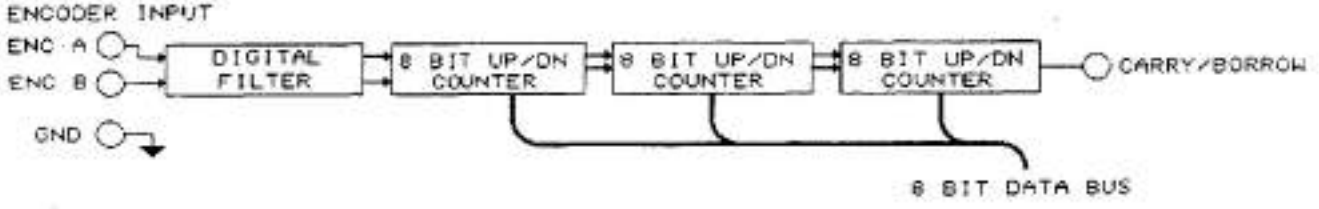


Figure 5-4. MSTEP5 Encoder counter interface (1 of 2 identical channels).

5.3 READING AND RESETTING THE ENCODER COUNTERS

The initialization mode 12 of the MSTEP.BIN driver hardware resets both encoder counter channels to zero. Other than performing a re-initialization (which also affects the PPMC-101C stepper controllers), the MSTEP.BIN driver software provides no other way of resetting the encoder counters. The MSTEP5 hardware does have this capability, although it was omitted from the driver in the interests of simplicity. Usually resetting the counters is unnecessary as it is easy to track relative changes in software, however if you do need to perform a reset of an encoder channel after initialization, this is how it is done:

For Channel A:

```
xxx00 BASE = &H300                          'base I/O address
xxx10 OUT BASE + 2, 0                       'clears A counter
```

For Channel B:

```
xxx00 BASE = &H300                          'base I/O address
xxx10 OUT BASE + 5, 0                       'clears B counter ]
```

You can also read either counter with INPs and OUTs although this is not half as easy as using mode 9 of the driver to do the job:

```
xxx00 BASE = &H300                          'base I/O address
xxx10 CT% = INP(BASE)                       'read control register
```

To select Counter A:

```
xxx20 OUT BASE, (CT% AND &HF7)              'clear ECTRAB bit
```

Or to select Counter B:

```
xxx20 OUT BASE, (CT% OR &H08)               'set ECTRAB bit
```

Then read selected counter:

```
xxx30 BL% = INP(BASE + 3)                   'low byte
xxx40 BM% = INP(BASE + 4)                   'middle byte
xxx50 BH% = INP(BASE + 5)                   'high byte
xxx60 CT# = BL% + BM%*256 + BH%*256^2'count
xxx70 OUT BASE, CT%                         'restore control if you want
```

The MSTEP5 board was designed mainly for move-stop-move-stop-etc. applications. As a result, to keep the hardware simple, the encoder counters were not latched and in most practical applications of the MSTEP5 board, this should not be a problem. Since each counter byte is read in turn, the data may change during the 3 byte read if the encoder is moving. There is a chance that you can return an erroneous count under these circumstances. There are 2 ways of avoiding this problem with the MSTEP.BIN driver:

1. If the encoder is mechanically coupled to an associated stepper motor, find out if the motor is at standstill using a MODE 8 status read before reading the encoder using MODE 9 (encoder read):

```
xxx00 MD% = 8                               'status read
xxx10 CALL MSTEP (MD%, D%(0), STP#, FLAG%)
xxx20 IF FLAG%=1 GOTO xxx10                 'wait till motor stops
xxx30 MD% = 9                               'encoder read
xxx40 CALL MSTEP (MD%, D%(0), STP#, FLAG%)
xxx50                                       'STP# contains encoder data
```

2. If the encoder is not coupled to a stepper motor input e.g. a manual input or may be moving slowly, read it twice in quick succession. If you get the same reading both times, it must be correct:

```
xxx00 MD% = 9                               'encoder read
xxx10 A# = 0                                'declare 2 double precision
xxx20 B# = 0                                'variables
xxx30 CALL MSTEP (MD%, D%(0), A#, FLAG%)
xxx40 CALL MSTEP (MD%, D%(0), B#, FLAG%)
xxx50 IF A#<>B# THEN GOTO xxx30             'try again
xxx60                                       'Either A# or B# holds correct data
```

This method will work fine as long as the encoder is moving slowly or is stopped. If the encoder is moving fast, you run the risk of getting stuck in a loop, but it is easy to add a timeout count to avoid this problem.

5.4 CONNECTING UP A SHAFT ENCODER

If you are using the STA-STEP, follow the instructions in the next section. If you want to connect directly to the MSTEP5, the encoder outputs A and B should be directly connected to the encoder A and B inputs (pins 35 & 36 for channel A, or pins 38 & 39 for channel B). Also make sure that the encoder common or ground is connected to one or more of the ground input pins of the MSTEP5 (pins 8,16,17,25,33,34,49 or 50).

The encoder will only draw a small current, usually less than 200mA which can be supplied from the computer. If you wish to do this, connect the encoder +5v supply to either pin 47 or 48 of the MSTEP5 connector. @B[If there is any possibility of frequent external short circuits of the encoder power to ground or high voltages, use a separate 5v power supply for the encoder]. It is better to destroy the encoder rather than the computer as well if you cannot be sure of the safety of the external wiring.

* * * * *

CHAPTER 6

USING THE STA-STEP

6.1 STA-STEP DESCRIPTION

The STA-STEP is designed to provide in a convenient external box, a dual channel isolated L/R type of stepper motor translator plus convenient screw connections for all the other MSTEP5 functions such as encoder channels, limit switches etc.

Each stepper motor driver consists of five power mosfet transistors driven through optocouplers so that the motor power circuit is totally isolated from the computer. Each of the power mosfets is driven by one of the phase outputs S1-S5 from the MSTEP5 board, and the corresponding power transistor output appears on W1-W5 (Winding 1 thru Winding 5). Depending on the programming of the PPMC-101C controllers in initialization (MODE 12) you can drive any of the following types of motors:

3 Winding - use W1, W2, and W3 (Motor Code Type 1)
4 Winding - use W1, W2, W3, and W4 (Motor Code Type 2)
5 Winding - use W1, W2, W3, W4, and W5 (Motor Code Type 3)

The power mosfets have ON resistances of 0.08 Ohm, 60V breakdown rating, and 25A continuous-drain current. On the closely spaced heatsinks of the STA-STEP, it is recommended that you limit the maximum current to 5A so that you can drive motors with ratings up to 24V and 5A per winding. The STA-STEP also includes locations for mounting current limit resistors. These are optional and speed up motor response (see next section and Section 6.2). If you use these resistors mounted on the STA-STEP board, limit each resistor dissipation to 5W or less, if the dissipation is higher, mount them external to the STA-STEP.

6.2 POWER DRIVE CIRCUIT

Each of the phase drives S1-S5 has a corresponding power output stage as shown below. This includes transient suppression (D1 and R1) which are required for switching inductive loads. Note that through the use of the optoisolator, all the motor power switching circuitry is totally isolated from the computer power and ground. One slight disadvantage of the optoisolator is that they have rise and fall times of several microseconds which imposes a limit on the maximum stepping rate. It is recommended that the rate be kept below 2,000 steps/second.

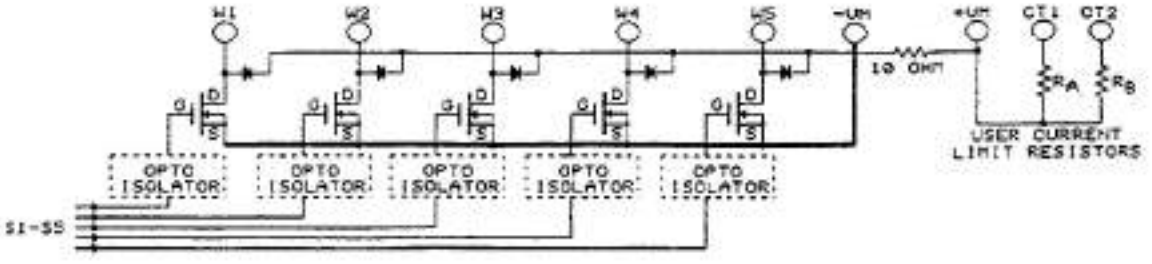


Figure 5-1. STA-STEP Power Drive Circuit.

In the ON state, the transistor power dissipation is controlled by the motor current and ON resistance of the power transistor. The on resistance of the power mosfet is about 0.08 Ohm, and it is recommended that motor current should not exceed 5A. This corresponds to a transistor power dissipation of 2W, which is comfortably handled by the heatsinks that the transistors are mounted on. In normal operation, these heatsinks should be barely warm. Switching losses will also account for a small amount of additional power loss. The drain voltage of the transistors is rated at 60V, this should be more than adequate for 99% of commercially available stepper motors. If you do need a higher voltage, contact MetraByte.

Due to the logic inversion in the power driver, the output logic polarity of the PPMC-101C phase drives should be set to inverted when using the STA-STEP. To do this, set D%(6)=0 in initialization mode 12.

6.3 STA-STEP CONNECTIONS TO STEPPER MOTORS

The typical connections of an STA-STEP to a Superior Electric stepper motor are shown in Figure 6-2. As explained in Section 6, use of additional resistance in series with the windings will reduce the winding time constant and improve the performance of the motor at high speed. The disadvantage to this arrangement is that you will waste power in the series resistors and require a higher motor supply voltage for a given maximum winding current. The STA-STEP includes space for these additional resistors at least for smaller stepper motors. It is recommended that the resistors which should be power wirewound types, **not** be mounted on the board if the power dissipation in each resistor will exceed 5 watts, mount them externally instead. If you are not going to use current limiting resistors, then a jumper can be placed in the corresponding location (RusrA & RusrB for channel A, RusrC & RusrD for channel B).

The motor power should be derived from an external power supply. A simple unregulated supply will usually be adequate. It is not recommended to attempt to run the motor from the computer +5V which is available on the STA-STEP, usually the load will be too great and the switching of the motors will generate considerable power supply noise. Note that the -VM (common) motor power supply sides of each channel are joined together on the STA-STEP, however, the +VM inputs are not common. This means you can use motors of different voltage ratings on each channel with separate supplies, or by externally jumpering the +VM inputs, you can use one supply to power two identical motors.

6.4 STA-STEP CONNECTIONS TO ENCODERS

The encoder output signals should be connected to ENA and ENB as shown in Figure 6-3. Also the encoder common should be connected to system ground. Unlike the stepper motors, there is no isolation between the encoder outputs and the computer ground and input signals. As the encoder is by its nature an optically isolated component, this should not present too much of a problem. The encoder may be powered from an external regulated 5v supply or as its power needs are low, it can also be powered from the computer 5v supply available on the STA-STEP connector. If you use computer power, take care not to abuse it with short-circuits to ground or other voltages - this can damage your computer.

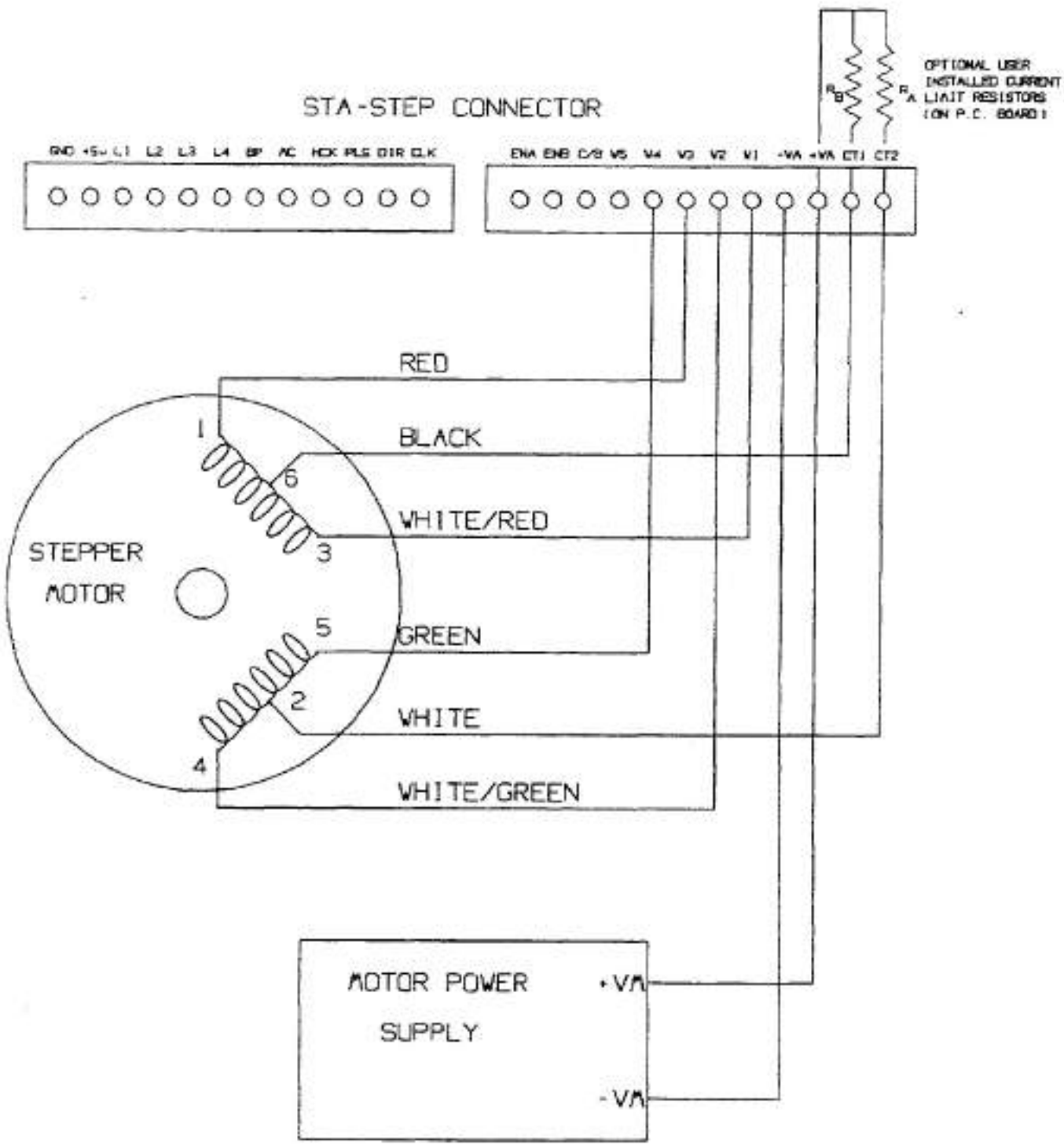


Figure 6-2. Connection of STA-STEP to STEP-MOT1 (or similar stepper motor).

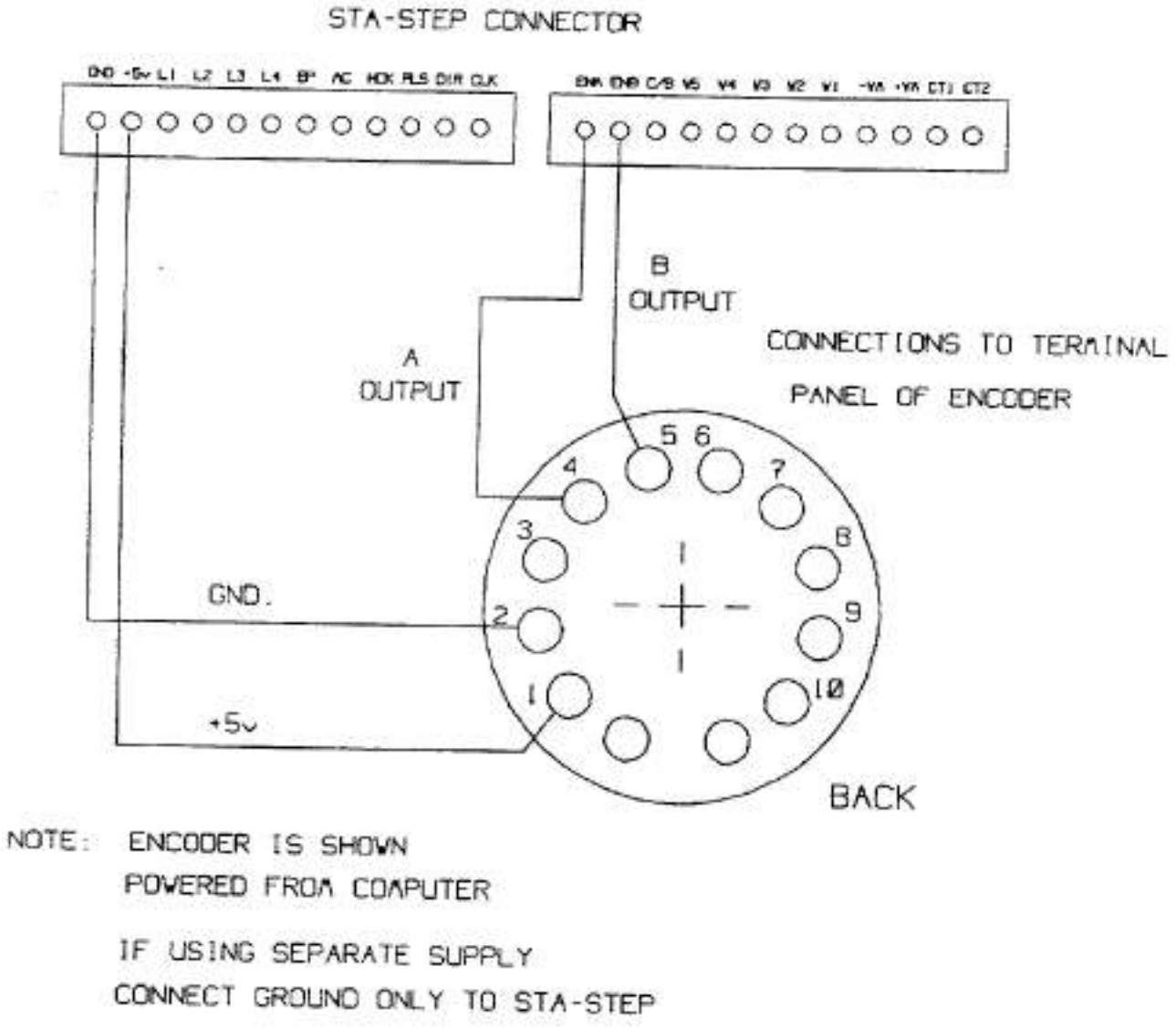


Figure 6-3. Encoder connections to STA-STEP.

6.5 CONNECTION TO EXTERNAL TRANSLATOR

If you prefer to use a commercially available translator rather than the power drive circuits of the STA-STEP, you can drive the translator from the DIR (CW/CCW) and PLS (pulse) output signals on the screw connector. Your translator should also share a common ground with the GND terminal of the STA-STEP, see Figure 5-4.

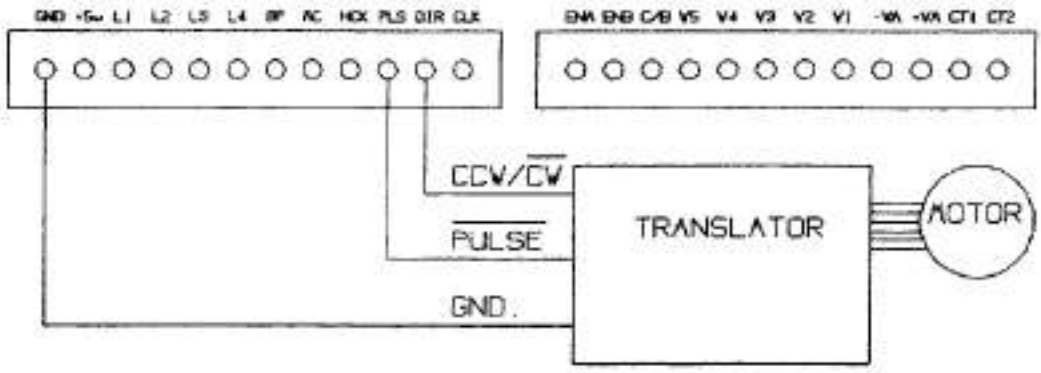


Figure 6-4. Connecting an external translator to STA-STEP.

The DIR and PLS signals are buffered outputs from the PPMC-101C controller. DIR is logic high for counter-clockwise direction and low for clockwise, PLS is a 5 microsecond negative going pulse for each step. These signals correspond to CCW/CW and P-OUT which are more fully described in the PPMC-101C data sheet in Appendix A.

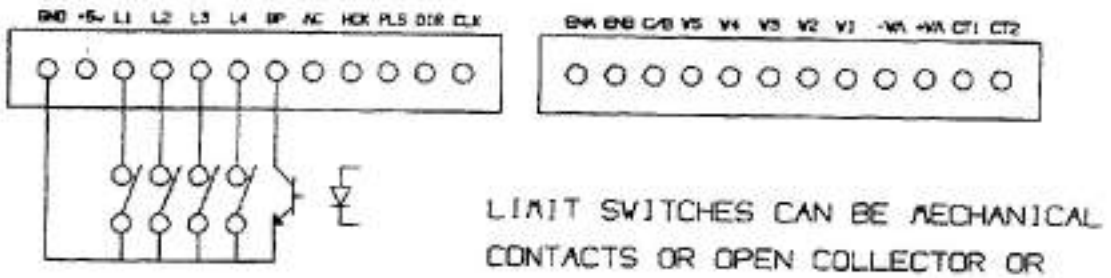
An external translator, particularly of the chopper type, may offer some improvements in efficiency and motor operating characteristics. If you are concerned about obtaining maximum performance and efficiency, an external translator such as Superior Electric's models 230T or 430T will be a better choice than the inexpensive L/R driver of the STA-STEP.

6.6 OTHER SIGNALS ON THE STA-STEP

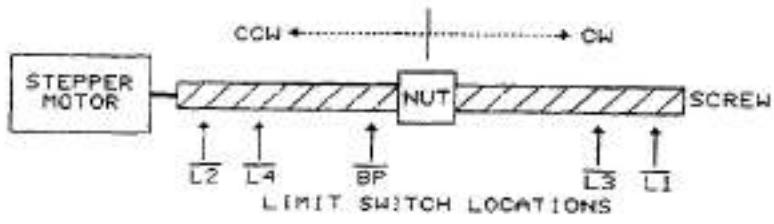
The STA-STEP also provides connections to the other remaining control signals of the MSTEP-5 board. All unused inputs can safely be left open circuit. These connections include:

L1, L2, L3, L4, & BP	Limit switch inputs. These include internal 10K pullups to +5V and are active low. A simple mechanical switch or open collector opto-interrupter between the limit switch input and ground is all that is needed, see Figure 6-5. An unused limit switch input can be left open circuit.
MC	Motor control. This input is the same as the MON (motor on) signal detailed in the PPMC-101C data in Appendix A. It has an internal 10K pullup to +5v. If taken low or connected to ground, it will suppress output pulses from the PPMC-101C. As it can be read from the status register, it can be used to monitor motor power if needed.

HCK	Hold acknowledge output corresponds to HOLD of the PPMC-101C data sheet in Appendix A. HCK is only active if you have selected switching at standstill on in initialization. HCK goes high 3 milliseconds after the motor stops, and will go low on receipt of the next motion command.
CLK	This is the external stepper clock source input (D%(7) = 2 in mode 12 initialization). Maximum input frequency should not exceed 133KHz for reliable operation (see CLOCK in Appendix A).
C/B	This is the carry/borrow output from the 24 bit incremental encoder counter. It can be used to extend the counter externally if needed.



Physical arrangement



All limit switch inputs are active low, they will have no effect if left open circuit.

L1 and L2 are overtravel limits and must usually be connected.

L3 and L4 are deceleration limits, their use is optional.

BP (Base point or CNP) is a home reference input. It can be a separate input or combined with L1 or L2 (connect BP to L1 or L2).

Figure 6-5. Limit Switch connections to STA-STEP.

CHAPTER 7

STEPPER MOTORS & TRANSLATORS

7.1 HOW A STEPPER MOTOR WORKS

The diagrams in Figure 7-1 will assist you in understanding how a stepper motor operates. In a real stepper motor there are are more poles on the stator and rotor, but the operation is similar. The rotor of the simplified version consists of a cylindrical magnet sandwiched between two discs each with 3 poles on them. The discs are rigidly clamped together and a half pole pitch out of step with each other (in this case 60 degrees out of alignment). Since the poles stick out (salient poles) on both the stator and the rotor, there are positions where the magnetic circuit or reluctance is at a minimum and the rotor will tend to align in these preferred positions even when the power is off. As soon as power is applied to the windings, this "locking" effect becomes even more pronounced, the external torque required to move the rotor to the next preferred position is known as the holding torque. The holding torque with power applied is usually about 10 times the holding torque with the stator de-energized.

By turning the windings on the stator on in sequence, it is possible to pull the rotor around in either direction according to the sequence. The diagrams show 4 steps of the motor, note how the windings are alternately energized and the current reversed to accomplish the stepping (remember like poles repel and unlike poles attract!). In this example each switching of the windings results in the rotor moving 30 degrees, so this motor would have a 30 degree step angle and take 12 steps to complete a revolution.

Most commercially available stepping motors have finer step angles of 1.8, 7.5 and 15 degrees with the 1.8 degree (or 200 step/rev) motor being one of the most popular. Smaller steps require more poles on the rotor and stator, in fact for 200 step/rev. motors, the rotor resembles two 50 tooth gearwheels although the stator uses some tricks by grooving the poles to reduce the number of windings and hence complexity. The permanent magnet of the rotor is magnetized to the maximum extent by the manufacturer within a magnetic circuit similar to the stator and cleverly assembled without removing the magnetic circuit. If the rotor is removed from the stator, the loss of the magnetic circuit will tend to self demagnetize the magnet. This leads to a loss of torque on re-assembly as the magnet is no longer as powerful. If you want to see what a real stepper motor looks like inside, bear this in mind before you tear a good motor apart. Some stepper motors do not use a permanently

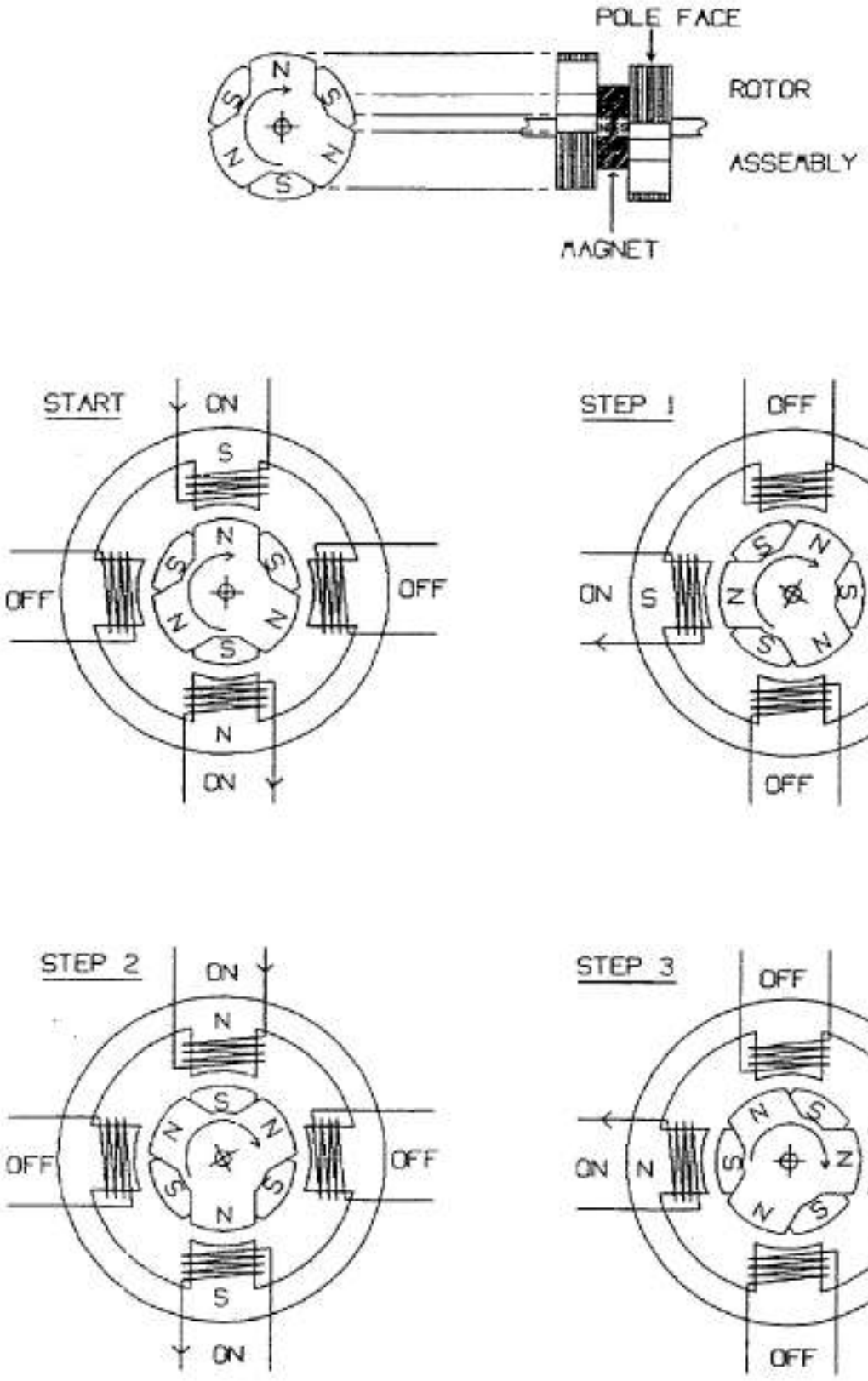


Figure 7-1. Stepper motor operation.

magnetized rotor, but operate similarly to those that do, except that they have zero holding torque when de-energized. Some stepper motor drives use a technique known as microstepping which leads to very small step angles of a few tenths of a degree. Instead of switching the windings completely on or off, the windings are energized partially so that the resultant magnetic field vector is moved in much smaller increments than the pole angles. This requires a much more complex drive circuit and a slightly different type of design of stepping motor. The MSTEP-5 cannot microstep motors directly, although it can provide the pulse and direction outputs for a microstepping translator.

7.2 TORQUE VERSUS SPEED

As the stepping rate or speed of rotation of a stepping motor rises, the torque that the motor can provide tends to fall. A typical torque/speed curve is shown in Figure 7-2.



Figure 7-2. Typical torque/speed curve of a stepping motor.

As the speed rises, the winding currents tend to reduce due mainly to two effects. In turn the reduced winding current leads to reduced torque. The two effects that cause this are as follows:

Back EMF — As the magnetized rotor turns within the stator, it induces a voltage in the windings which opposes the driving voltage. This in turn reduces the winding current. The induced EMF is proportional to speed, rising as the speed increases. The solution to this is to increase the driving voltage as the speed rises, but this greatly increases the complexity of the drive electronics.

Inductance — The winding inductance and resistance control the rate of rise of current in accordance with the well known relation

$$I = (1 - E^{-Rt/l}) * V/R$$

The quantity L/R is known as the time constant, the smaller it is, the faster the current will rise on switching the winding on. Plainly we need the smallest inductance and largest resistance to reduce the time constant although this would lead to a very lossy and inefficient winding. In practice we can add a bit of external resistance and improve the performance, although the resistor wastes power. This forms the basis of the simple L/R translator as embodied in the design of the STA-STEP.

7.3 RESONANCE

Each time the rotor is stepped, it tends to overshoot the desired position slightly and performs a small damped oscillation. The damping is mainly electrical due to induced voltage in the windings. The frequency of the damped oscillation depends on the total inertia of the system. It is possible that at certain speed the stepping rate will become a harmonic of the natural oscillation frequency, and under these conditions the oscillation builds up to a point that the motor misses steps. This condition is known as resonance.

Resonance problems can be avoided by

1. Not operating in or close to a resonance region.
2. Accelerating the motor through the resonant point as fast as possible.
3. Providing additional damping in the form of mechanical or viscous damping (Lanchester dampers) and/or altering the moment of inertia.

If the motor is operated at low stepping rates, resonance is not usually likely to be a problem.

7.4 FULL- AND HALF-STEP OPERATION

There are two common switching sequences used for stepper motors. In the full-step sequence, two windings are always energized. This applies whether the motor has three, four, or five phase windings. For example, for a 4-phase motor

FULL STEP:

	WINDING 1	WINDING 2	WINDING 3	WINDING 4
Step 1	on	on	off	off
Step 2	off	on	on	off
Step 3	off	off	on	on
Step 4	on	off	off	on
etc...				

Full step gives the greatest torque and the rated step angle of the motor e.g. 200 steps/rev.

In half step sequence, switching alternates between having 2 windings on and 1 winding on. It takes twice as many steps to travel the same distance, thus a 200 step/rev. motor with 1.8 degree stepping angle in full step sequence will produce 400 steps/rev. and 0.9 degree step angles in half step sequence.

HALF STEP:

	WINDING 1	WINDING 2	WINDING 3	WINDING 4
Step 1	on	off	off	off
Step 2	on	on	off	off
Step 3	off	on	off	off
Step 4	off	on	on	off
Step 5	off	off	on	off

HALF STEP:

	WINDING 1	WINDING 2	WINDING 3	WINDING 4
Step 1	on	off	off	off
Step 2	on	on	off	off
Step 3	off	on	off	off
Step 4	off	on	on	off
Step 5	off	off	on	off
Step 6	off	off	on	on
Step 7	off	off	off	on
Step 8	on	off	off	on
etc...				

Half step sequence offers a finer positioning resolution and somewhat smoother stepping, but since fewer windings are excited on average, the torque and maximum speed capabilities can be lower than full step. The PPMC-101C can operate in either sequence according to the initialization parameters in mode 12.

7.5 TRANSLATORS & INDEXERS

If you are new to stepper motors, you will start reading catalogs and find all sorts of equipment with strange names. This little explanation may assist you in understanding some of the terminology and its relevance to the MSTEP-5.

A TRANSLATOR is a device that takes a step pulse and direction input and drives the stepper motor windings - it is essentially the power drive and pulse pattern determining stage, but it does not keep count of the pulses or position. An INDEXER goes further and will take an input from switches or a keyboard and move the motor the number of steps input at the keyboard, it also keeps track of the motor position.

In this sense, the PPMC-101C and STA-STEP form a TRANSLATOR, whereas an IBM PC, MSTEP-5 and STA-STEP and a suitable program can perform functions equivalent to an INDEXER.

The MSTEP-5 does not have to be used with the STA-STEP. It can just as easily be connected to a commercial TRANSLATOR using the direction and pulse outputs of each channel. Commercial INDEXERS are usually complete self-contained positioning systems and cannot be connected to the MSTEP-5.

7.6 MECHANICAL DESIGN

Simple systems can often be assembled with very little involvement in optimizing the mechanical design. They can almost be put together on a "try it and see" basis, this is one of the advantages of the simplicity of stepper motors. On the other hand, if you want to obtain maximum performance especially with larger drives, you should carefully analyze your mechanical requirements and match the motor and translator to your needs. Torque and power of stepper motors depend on the motor frame size and rating and range from 35 oz/in and fractional H.P. to 5000 oz/in and ratings of 3-4 H.P.

Other considerations are holding torque, motor size and design - shafts are available on one end or both ends etc. If you need leadscrews, X-Y tables etc. many vendors provide this type of accessory equipment, a short list is provided in Appendix B.

* * * * *

CHAPTER 8

TESTING & MAINTENANCE

No periodic calibration or maintenance is required for the MSTEP-5 or STA-STEP, there are no user adjustments.

The best method of testing a system is to run the DEMO program and exercise the motor or shaft encoder using this program. If you receive error code 5 while running the demo program, there are a number of possible conditions to check before assuming that the MSTEP-5 board is faulty, as follows:

1. If you receive error #5 on initialization of both channels, it means that the driver software is not locating the MSTEP-5 at the I/O address specified in the initializing parameters. This may be caused by the BASE I/O ADDRESS dipswitch being set to an address other than that specified, a conflict with the address of another peripheral or a truly faulty MSTEP-5. The driver software performs a quick write/read test to the MSTEP-5 control register on initialization, and if it detects any discrepancies will return error #5.
2. If you consistently receive error #5 on initializing one of the two channels, or receive it in the motion commands, it points to a faulty PPMC-101C controller. These are plug in chips and can be replaced. A PPMC-101C which is not correctly handshaking with the driver when it is ready to receive or provide data will give this hardware fault.

Other errors that are possible are open limit switch wiring, faulty limit switches etc. These can be checked by running the READ STATUS option of the DEMO program and exercising the limit switches.

If you wire the stepper motor up incorrectly with its windings in the wrong sequence, it can lead to bumpy or rough operation or dithering and similar effects. Check the stepper motor wiring and make sure that the windings are switching on and off using a voltmeter and the JOG command.

The DEMO program will expose 99% of the likely problems that you can have and also provides a baseline for comparison if you suspect you have a programming problem. The remaining 1% of hardware problems, stuck bits, open lines etc. can be determined by writing and reading the I/O ports. This tends to be tedious work and a program such as DEBUG which makes I/O operations short and simple is the best choice for this chore.

If you have problems with your MSTEP-5, please call MetraByte applications engineering at (617)-880-3000. If you need to return any hardware, we will issue you an R.M.A. (return material authorization) number to mark on your package. Please do not send us material without an R.M.A. number as it greatly complicates tracing its origins and faults. It is also useful to our test technicians if you can include a brief description of the problem and in what circumstances it occurs.

* * * * *

CHAPTER 9

INSTRUCTIONS FOR PRODUCT RETURN

Before returning any equipment for repair, please call 508/880-3000 to notify MetraByte's technical service personnel. If possible, a technical representative will diagnose and resolve your problem by telephone. If a telephone resolution is not possible, the technical representative will issue you a Return Material Authorization (RMA) number and ask you to return the equipment. Please reference the RMA number in any documentation regarding the equipment and on the outside of the shipping container.

Note that if you are submitting your equipment for repair under warranty, you must furnish the invoice number and date of purchase.

When returning equipment for repair, please include the following information:

1. Your name, address, and telephone number.
2. The invoice number and date of equipment purchase.
3. A description of the problem or its symptoms.

Repackage the equipment. Handle it with ground protection; use its original anti-static wrapping, if possible.

Ship the equipment to

Repair Department
Keithley MetraByte Corporation
440 Myles Standish Boulevard
Taunton, Massachusetts 02780

Telephone 508/880-3000
Telex 503989
FAX 508/880-0179

Be sure to reference the RMA number on the outside of the package!

* * * * *

APPENDIX **A**

PPMC-101C SPECIFICATIONS & PROGRAMMING

The data sheet of the stepper motor controller chip used in the MSTEP-5 is reproduced in its entirety here by permission of Sil-Walker America [Sil-Walker America Inc., 653 Las Casas Avenue, Pacific Palisades, CA. 90272 Phone:(213)-454-4772]. If you intend programming the stepper channels directly, want to understand how the MSTEP.BIN driver works, write your own routines, or know more about the limit switch operation or fine details of the controllers, these data pages will provide most of the answers.

Due to the mapping of the PPMC-101C registers into the I/O map of the MSTEP-5 the following correspondences apply:

A0RSEL of the MSTEP-5 control register = A0 of both PPMC-101C
CSA of the MSTEP-5 control register = CS of channel A PPMC-101C
CSB of the MSTEP-5 control register = CS of channel B PPMC-101C

CSA and CSB should never be low together as the 2 controllers will interfere with each other.

RESET of either PPMC-101C is accomplished on the MSTEP-5 either by the active RESET line of the IBM PC bus (on power up) or from a monostable triggered by a write to a I/O address BASE + 3 for channel A, or BASE + 4 for channel B. In this way the programmer has control of hardware resetting and re-initializing the controllers if needed.

The PPMC-101C has a couple of minor programming quirks to watch. The motion commands accelerate/decelerate or move at constant speed will always step one pulse more than the number input. The MSTEP.BIN driver corrects for this characteristic in these two commands by subtracting one from the data before sending it along to the PPMC-101C's. Also the singular data values 0 and 1 are caught in the driver - see MSTEP.ASM. If zero data is applied, the PPMC-101C will step 1 pulse, whereas if 1 is supplied it will step 2 pulses. The MSTEP.BIN driver grabs these two conditions and does nothing if zero is applied or aborts to a jog command if 1 is applied. In this way the MSTEP.BIN driver always does what you would expect - the PPMC-101C has somewhat different although consistent rules.

INTRODUCTION

PPMC101C/102A is a unique one-chip LSI specially designed to interface a stepper motor to an 8-bit micro computer with no additional hardware. PPMC101C/102A provides 8 kinds of different operations by the command of master CPU including acceleration/deceleration and constant speed operation.

Operating frequency and number of phase for stepper motor are programmable. Distribution signal to excitation driving circuit can also be programmble for selection of 2-phase or 1-2 phase excitation (2-3 phase excitation for 5-phase motor) for 3, 4 and 5 phase motor.

In addition, PPMC101C/102A provides five kinds of "limit" switch input. Complete function necessary to control stepper motor is included in one chip LSI. The PPMC101C/102A can be easily interfaced with a microcomputer system.

1. PPMC101C/102A SPECIFICATIONS

Operation Command

- *Emergency Stop*
- *Decelerating Stop*
- *Single Step*
- *Acceleration & Deceleration*
- *Constant Speed Operation*
- *To move to the "Limit"*
  - *(1) To move to the high speed limit*
  - *(2) To move to the base point*

Excitation Method

Motor	*Excitation*
3-phase	*2 phase*
	1-2 phase
4-phase	*2 phase*
	1-2 phase
5-phase	*2 phase*
	2-3 phase

Number of steps : 16,777,216 max

Number of pulse for acceleration/deceleration : 4 - 8,160

Maximum pulse rate : PPMC101C ... 5K pps (RA=20, fo=100KHz)
PPMC102A ... 10K pps (RA=20, fo=200KHz)

Power supply : 5V ± 5% 125mA max

2. TERMINAL ASSIGNMENT AND FUNCTIONS



(Top View)

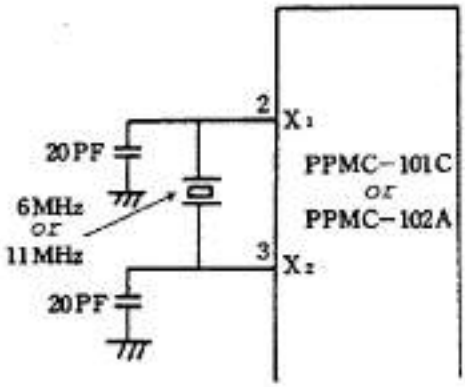
40 pin Dual-In-Line

Signal	Pin#	I/O	Description
X1, X2	2, 3	I	X-tal
$\overline{RESET}$	4	I	RESET input
$\overline{CS}$	6	I	Chip Select
$\overline{RD}$	8	I	READ strobe
Ao	9	I	Address 0
$\overline{WR}$	10	I	WRITE strobe
SYNC	11	O	Timing output
Do - D7	12-19	I/O	Data Bus 8-bit
$\overline{L4}$	21	I	Reverse high speed limit input
$\overline{L3}$	22	I	Forward " " " "
$\overline{L2}$	23	I	Reverse limit input
$\overline{L1}$	24	I	Forward " "
$\overline{P-OUT}$	27	O	Pulse output
CCW/$\overline{CW}$	28	O	Forward/Reverse status '0' = Forward '1' = Reverse
HOLD	29	O	Motor HOLD output
S5	30	O	Motor 5th phase output
S4	31	O	" 4th " "
S3	32	O	" 3rd " "
S2	33	O	" 2nd " "
S1	34	O	" 1st " "
$\overline{INT}$	35	O	Interrupt signal
MON	37	I	External control '0' = Motor ON '1' = Motor OFF
$\overline{CNP}$	38	I	Base point signal input
CLOCK	39	I	External clock input
Vcc	26,40	I	+5V DC
GND	7,20	I	0 V
NC1	1,5,36	I	pull up to Vcc with 3.3K ohm or open
NC2	25	O	OPEN

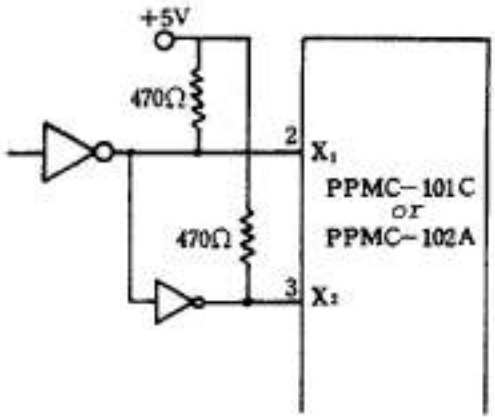
PIN DESCRIPTION

2-1 X1, X2

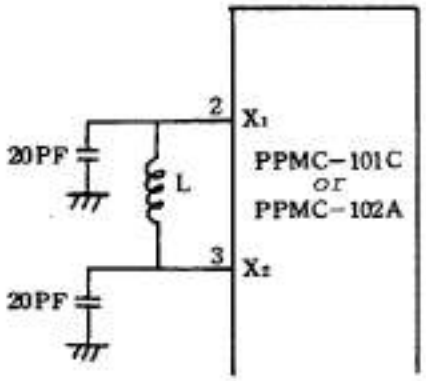
Inputs for a crystal. PPMC101C is normally operated with a 6 MHz crystal and PPMC102A with a 11 MHz crystal. You may also drive the clock inputs with an LC turned circuit or an external clock source (2 phases) as shown in Fig 2-1. 1 to 6 MHz for PPMC101C and 1 to 11 MHz for PPMC102A can also be used for driving frequency, but the operating speed slows down in accordance with the clock frequency.



X-tal Clock Driver



1 - 6 MHz Input Frequency External Clock Driver Circuit



LC Turned Circuit Clock Driver

Fig 2-1

PPMC-101C	PPMC-102A
L=130 μH 3 MHz	L=120 μH 3.2 MHz
L= 40 μH 5 MHz	L= 45 μH 5.2 MHz

2-2 $\overline{\text{RESET}}$

Input used to reset status flip-flops and to set the program counter to zero. This pin should be connected to the RESET signal of a user's system. 50msec after the RESET signal rising edge the PPMC101C/102A is operative for initialization and operation command. The pulse width of the RESET signal must be no less than 2.5 μsec.

2-3 $\overline{\text{CS}}$

Input for chip select. To input the decorded signal from upper bits of ADDRESS. PPMC101C/102A is accessible at a low level 'O' on CS.

2-4 $\overline{\text{RD}}$

I/O read input which enables the master CPU to write data and status words from the PPMC101C/102A. The OUTPUT DATA BUS BUFFER or status register can be READ at a low level 'O' on $\overline{CS}$ and $\overline{RD}$.

2-5 $\overline{\text{WR}}$

I/O write input which enables the master CPU to write data and commands words to the PPMC101C/102A. Data on INPUT DATA BUS BUFFER can be written at a low level 'O' on $\overline{CS}$ and $\overline{RD}$.

2-6 AO

Address input used by the master processor to indicate whether the byte transfer is data or command as shown in the following table

Ao	$\overline{RD}$	$\overline{WR}$
0	Data Resister	Data Resister
1	Status Resister	Command Resister

Table 1

2-7 SYNC

Output signal which occurs once per execution of internal command in the PPMC101C/102A. It is also used to synchronize the single step operation. It is to be normally OPEN and used to check IC operation.

2-8 DO-D7

Tri-state, bidirectional DATA BUS BUFFER lines used to interface the PPMC101C/102A to an 8-bit master system data bus.

2-9 $\overline{L1}$ – $\overline{L4}$, $\overline{CNP}$

Inputs for external 'Limit' switches. Each signal is activated at a low level 'O'. Fig 2-2 shows the idea of 'Limit" switches.

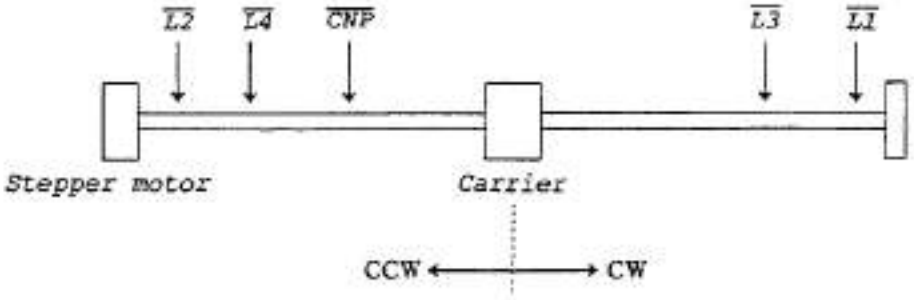


Fig 2-2

L1, L2 :

These switches are set at a maximum limit position where the carrier does not move further in CCW or CW. The motor will stop immediately when the carrier moves to these points regardless of the operation command. The carrier will no longer move further in the same direction even when it receives a command to move in the same direction. The carrier will start to move in the opposite direction only when it receives the command to move in reverse.

L3, L4 :

These switches must be positioned between L1 and L2, at a minimum distance corresponding to the number of deceleration steps. The stepper motor begins to decelerate at these positions (L3 or L4) in order to stop inside of L1 or L2.

$\overline{CNP}$:

Signal from $\overline{CNP}$ is used to establish a convenient reference point (Base point) with which the PPMC can monitor the position of the carrier. It does this by counting the number of steps in the data register. For example, in figure 2-2, in order to establish a convenient base point the command "move to base point" is used (see section 3-3-8). The motor will move the carrier to the position marked $\overline{CNP}$ and stop. Work can then proceed from this point.

2-10 $\overline{\text{P-OUT}}$, CCW/$\overline{\text{CW}}$

P-OUT is used for pulse output for other stepper motor driving modules without using PPMC phase output. It is useful for bipolar drive, switching drive and other special type of excitation method. It is recommended to use a decoder for CW or CCW pulse generation in combination with one-shot TIMER as shown in Fig 2-3 because driving module sometimes require 10 to 20 μsec pulse width. The signal from pulse output is a 5 μsec negative pulse and signal for direction is indicated by its level. In addition, these signals can be used for monitoring direction or number of pulses for rotation.

CCW/$\overline{\text{CW}}$ can be activated only when $\overline{\text{P-OUT}}$ is active.

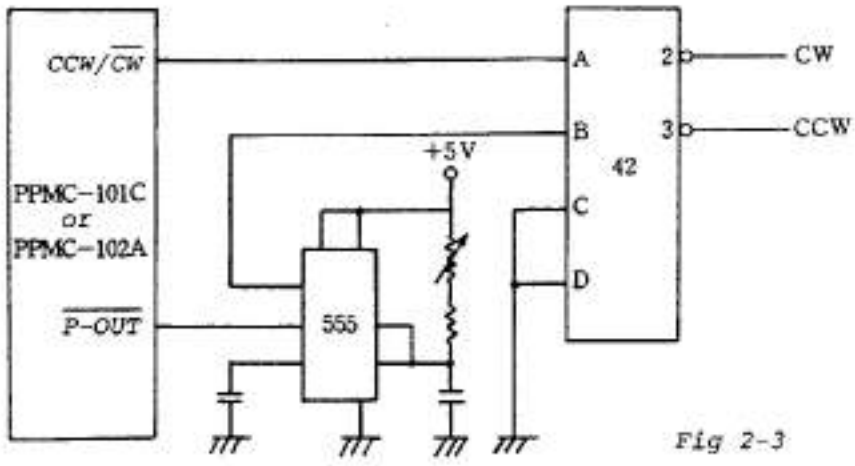


Fig 2-3

P-OUT is always available as well, in use of any type of phase output. (see page 13. motor code 01, 10, 11)

2-11 HOLD

HOLD output is high 3 msec after motor stops, but it is active only when bit 5 of the initialization command is set. (see page 13)

2-12 S1-S5

Provides signal for motor excitation drive.

Motor	Control
3 phase by	S1 - S3
4 " "	S1 - S4
5 " "	S1 - S5

Fig 2-4 shows the form of output.

The logic can be interchanged, positive to negative logic, and visa versa. Typical circuit is shown in Fig 2-5.

EXCITATION PULSE OUTPUT

2-phase excitation

	S1	S2	S3
1	▨	▨	
2		▨	▨
3	▨		▨

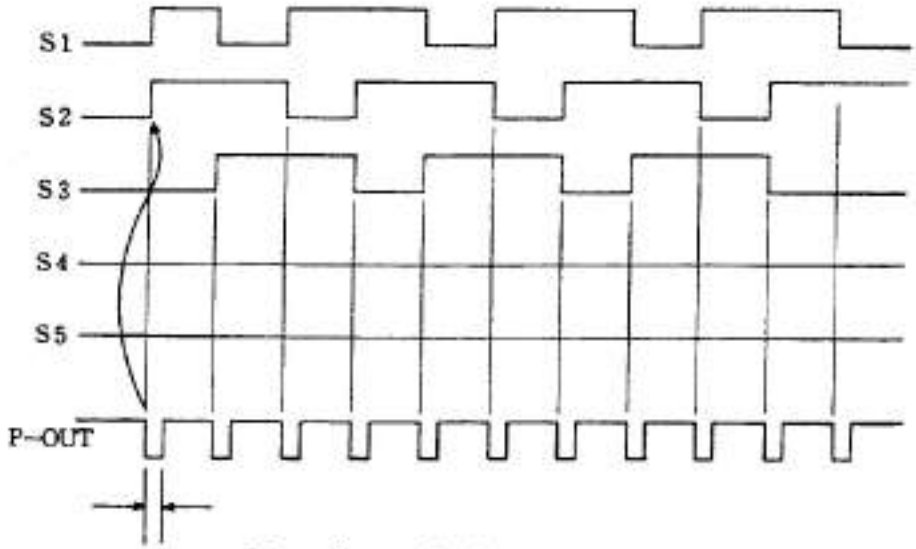
(3-phase motor)

	S1	S2	S3	S4
1	▨	▨		
2		▨	▨	
3			▨	▨
4	▨			▨

(4-phase motor)

	S1	S2	S3	S4	S5
1	▨	▨			
2		▨	▨		
3			▨	▨	
4				▨	▨
5	▨				▨

(5-phase motor)



Example - 3 phase motor, 2-phase excitation positive logic

Fig 2-4

1-2 phase excitation

	S1	S2	S3
1	▨		
2	▨	▨	
3		▨	
4		▨	▨
5			▨
6	▨		▨

(3-phase motor)

	S1	S2	S3	S4
1	▨			
2	▨	▨		
3		▨		
4		▨	▨	
5			▨	
6			▨	▨
7				▨
8	▨			▨

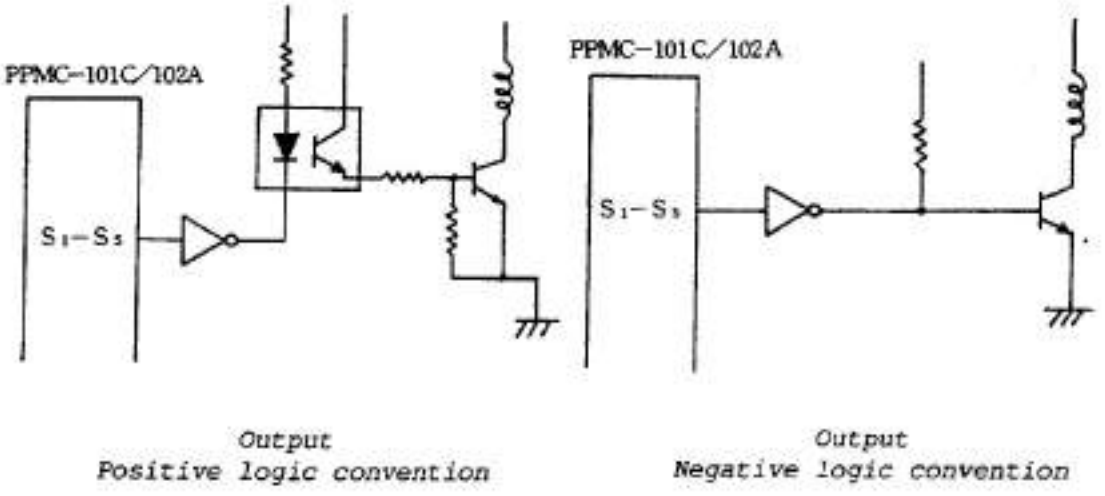
(4-phase motor)

2-3 phase excitation

	S1	S2	S3	S4	S5
1	▨	▨			
2	▨	▨	▨		
3		▨	▨		
4		▨	▨	▨	
5			▨	▨	
6			▨	▨	▨
7				▨	▨
8	▨			▨	▨
9	▨				▨
10	▨	▨			▨

(5-phase motor)

Output logic level can be switched by using positive or negative logic as shown in Fig. 2-5.



Output
Positive logic convention

Output
Negative logic convention

Fig. 2-5

2-13 $\overline{INT}$

Interrupt request is assertive 'O' when motor stops. $\overline{INT}$ can be cleared by reading the finish STATUS. This figure is not an open collector and OPEN COLLECTOR BUFFER is required as shown in Fig 2-6, when a multiple INTERUPT is expected.

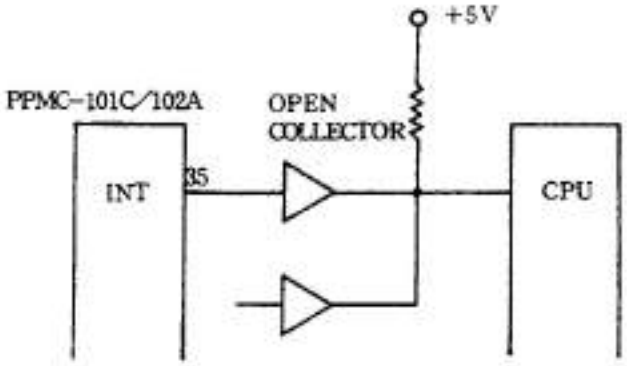


Fig 2-6

2-14 MON

When motor on input is 'O', PPMC does not output driving pulse. An example of an application is indicated in Fig 2-7, in which a thermal relay on the motor is used to protect overheating. MON input is ignored during operation of PPMC, and should be checked only before motor operation.

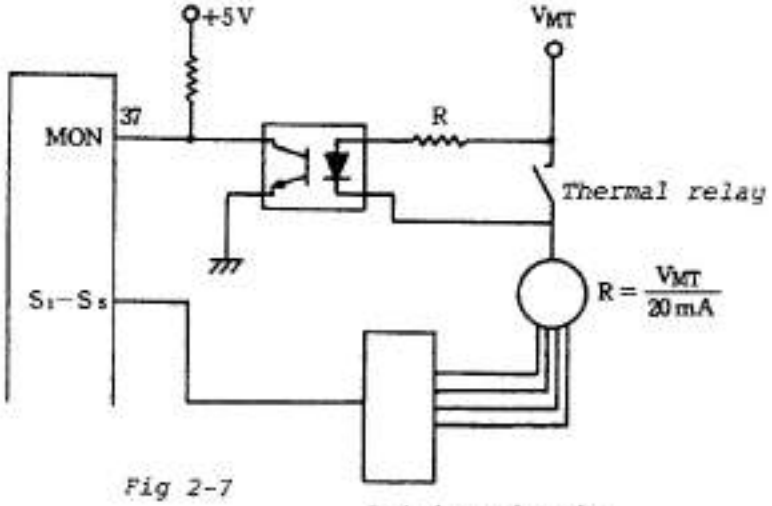


Fig 2-7

Driving circuit

2-15 CLOCK-external clock

Basic signal to control speed of the stepper motor. The speed can be controlled between 400pps and 5K pps by the 100KHz clock input to PPMC101C and between 800pps and 10K pps by 200KHz clock input to PPMC102A. The clock frequency must be below 1/45 of X1, X2 clock. For example, when 6MHz X-tal is applied for X1 and X2, external clock input must be less than 133KHz (in case 11MHz is applied, external clock must be less than 244KHz). High level of the clock pulse must be more than 500 nsec.(250 nsec for 11MHz)

3. COMMUNICATION BETWEEN PPMC AND MASTER CPU

The communication between PPMC and master CPU consists of following 3 types of modes :

(1) Initialization

It designates type of motor, method of excitation, data for acceleration/deceleration and other parameters (see page 13 for details). After power 'ON', initialization is needed before operation command. Note : Some parameters cannot be changed once it is set. Re-initialization is not possible during operation.

(2) Operation Command

8 kinds of operation commands are available for stepper motor. The length of data to follow depends on the command.

(3) Register for PPMC

After completion of (2), master CPU reads the cause of operation finish, status of input/output terminal, and the number of remaining pulse.

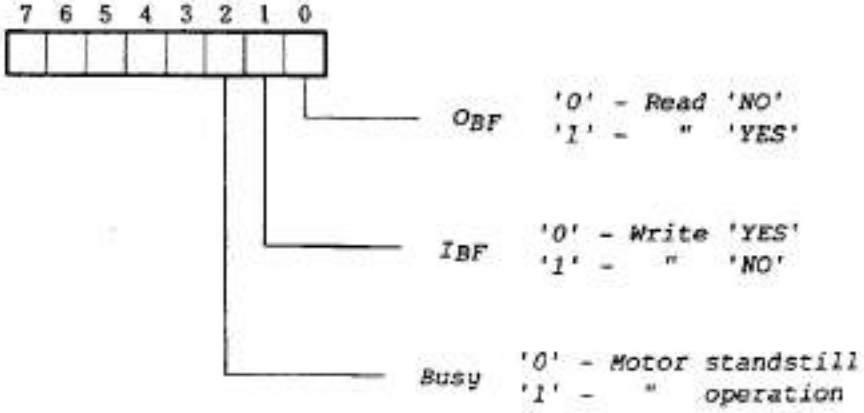
3-1 Register for PPMC

2 read only registers, and 2 write only registers are accessible to the uses.

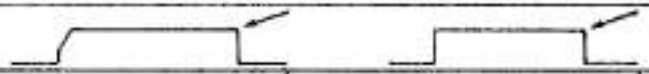
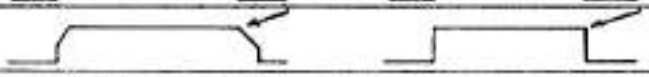
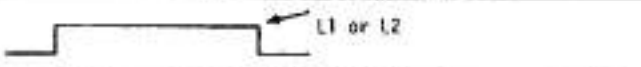
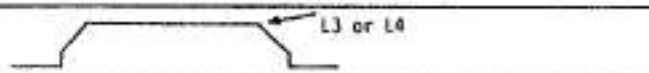
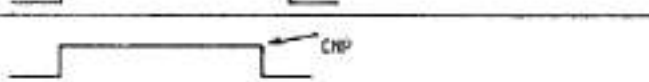
$\overline{CS}$	A_0	$\overline{RD}$	$\overline{WR}$	
1	×	×	×	DISABLE
0	0	0	1	READ DATA
0	1	0	1	READ STATUS
0	0	1	0	WRITE DATA
0	1	1	0	WRITE COMMAND

Table 3-1

3-1-1 Status Register



COMMAND TABLE

			COMMAND DATA	FUNCTION
INITIALIZATION		1 2 3 4 5	0 0 Self-starting pulse rate High speed pulse rate Accelerating/Decelerating pulse rate	
OPERATING COMMAND	Emergency Stop	1	0 1 0 0 0	
	Decelerating Stop	1	0 1 0 0 1	
	Single Step	1	0 1 0 1 0	
	Acceleration/ Deceleration	1 2 3 4	0 1 0 1 1 Number of operating step	
	Constant speed operation	1 2 3 4 5	0 1 1 0 0 Constant speed pulse rate Number of operating step	
	To move until the limit at constant speed	1 2	0 1 1 0 1 Constant speed pulse rate	
	To move until high speed limit	1	0 1 1 1 0	
	To move to the base point	1 2	0 1 1 1 1 Contant speed pulse rate	
READ REGISTER	Finish Data	1	1 0 0 0 0 0 0 0	To read data for reason of FINISH, etc. ..... 1 byte
	Input signal	1	1 0 0 0 0 0 0 1	To read data for limit switch, etc. ..... 1 byte
	Output signal	1	1 0 0 0 0 0 1 0	To read data for motor phase output and direction ... 1 byte
	Remaining step numbers	1	1 0 0 0 0 0 1 1	To read remaining number of steps ..... 3 byte

3-1-1-1 OBF (Output Buffer Full)

This bit checks the status in order to read the data from PPMC. 'O' indicates that there is no data in the buffer. It can only read the data when OBF is '1'.

3-1-1-2 IBF (Input Buffer Full)

This bit checks the status in order to write commands or input data to PPMC. '1' indicates that the data is full in the buffer and therefore, it is not possible to write new data. IBF must be 'O' when you write data or give commands. If you were to write data at IBF '1' the former data would be erased.

3-1-1-3 BUSY (Motor Busy)

This flag outputs '1' during motor operation. It is only possible to input emergency stop and decelerating stop commands at that time. The IBF and BUSY bits must be checked before you input a command. This is 'O' 2.5 µsec after INT output.

3-1-2 Read Register Data

Register data can be read out after checking OBF and input of READ REGISTER COMMAND.

3-1-3 Write Command

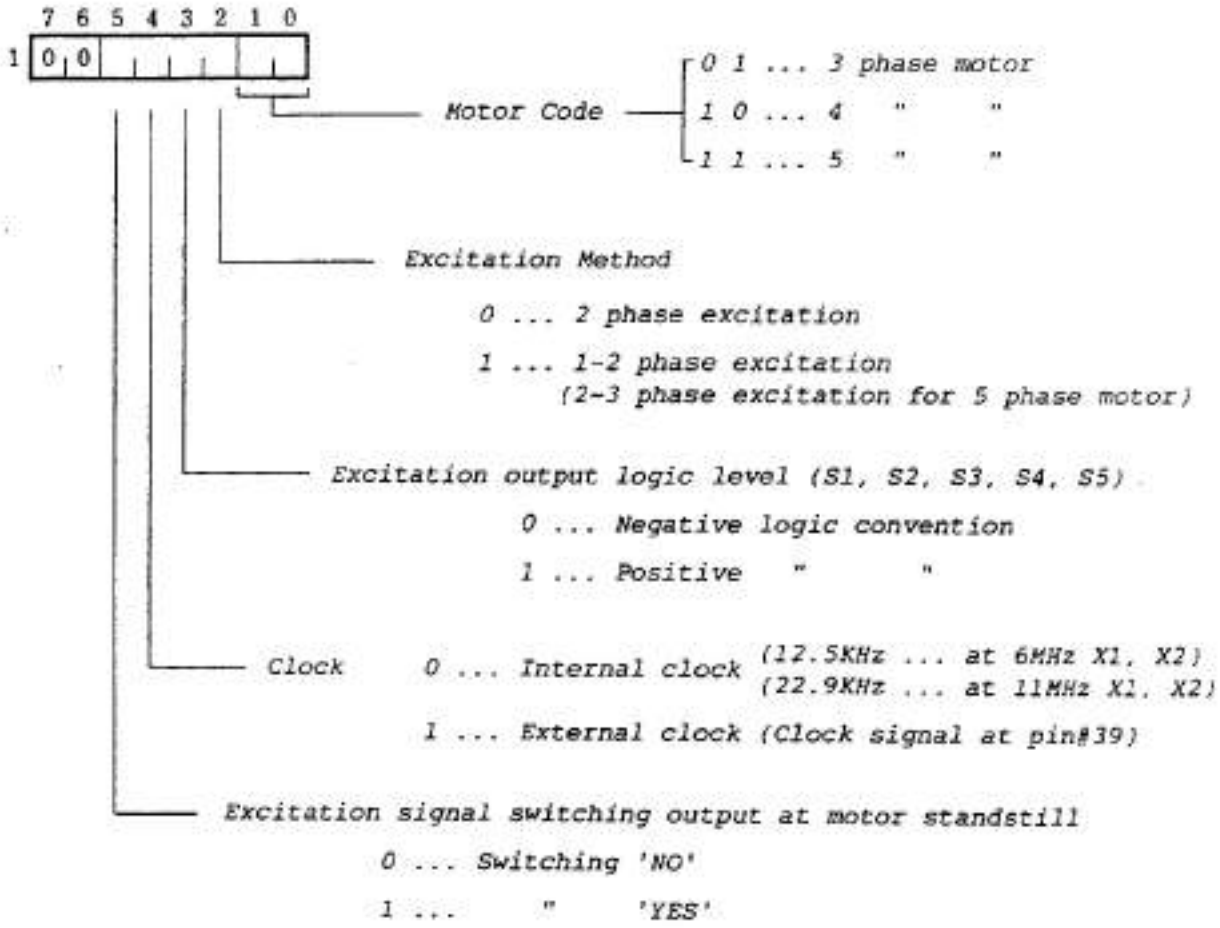
Before inputting initialization, operation command or read register command, check IBF and BUSY bit in the status register.

3-1-4 Write Data

Check IBF before writing data for pulse rate or number of steps. The order of input data must follow as indicated in command table (page 14). PPMC101C will start operating in accordance with the command 400 µsec and PPMC102A does 200 µsec after the data is written.

3-2 Initialization

Initialization Command



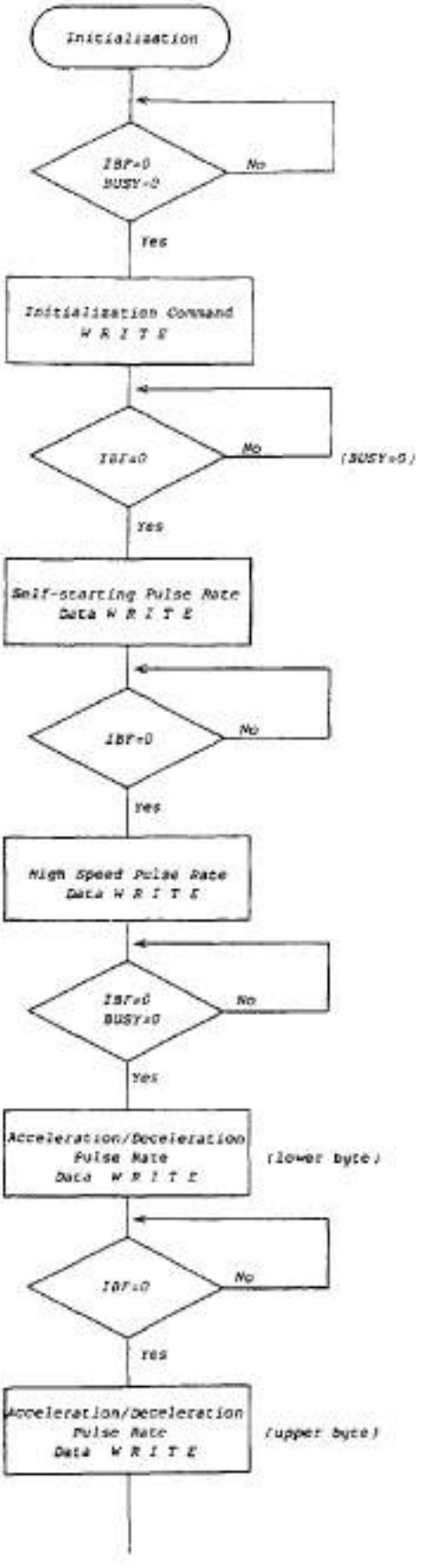
Initialization Data



self-starting pulse rate ..... RA max
high speed pulse rate ..... RA min
Acc/Deceleration pulse rate .... lower byte
" " " .... upper byte

Initialization command to be input in the above order (1, 2, 3, 4, 5) after power 'ON'.

It can be shown in the following flow chart.


Initialization
IBF=0
BUSY=0
No
Yes
Initialization Command
W R I T E
IBF=0
No
(BUSY=0)
Yes
Self-starting Pulse Rate
Data W R I T E
IBF=0
No
Yes
High Speed Pulse Rate
Data W R I T E
IBF=0
BUSY=0
No
Yes
Acceleration/Deceleration
Pulse Rate
Data W R I T E
(lower byte)
IBF=0
No
Yes
Acceleration/Deceleration
Pulse Rate
Data W R I T E
(upper byte)

3-2-1 Initialization Command

1) Motor code

The type of motor code used must match the spec of the motor.

2) Excitation method

The excitaion method used must match the spec of the motor.

3) Logic level of excitation output (S1, S2, S3, S4, S5)

In positive logic convention, the current will be flowing through the coil of the motor when output of PPMC is high. In negative logic convention, the current will be flowing through the coil of the motor when output of PPMC is low.

Fig 3-1 shows the circuits of logic level of excitation output.

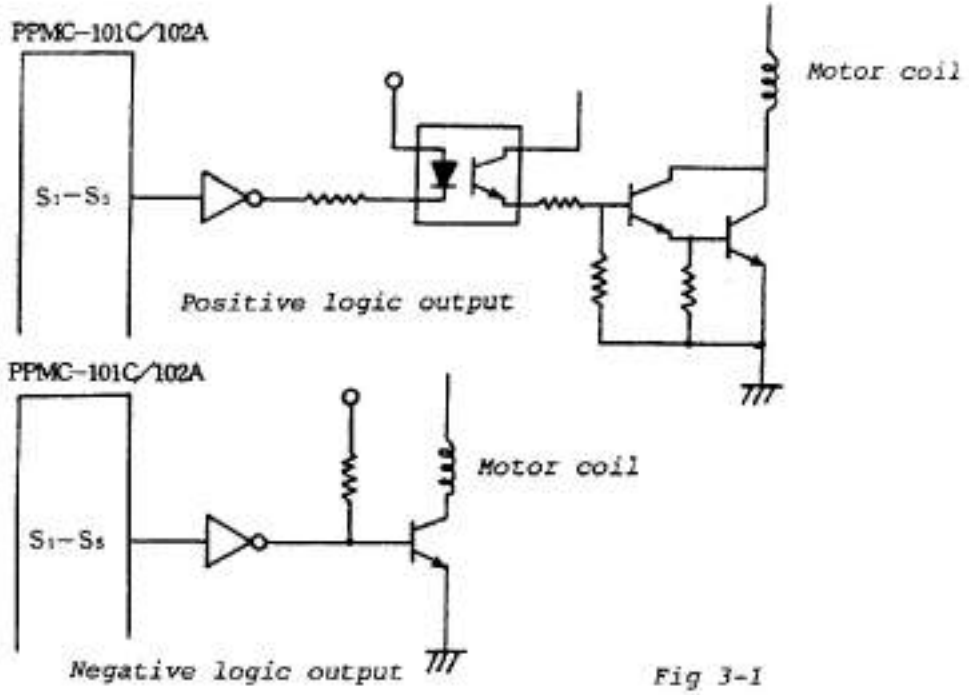


Fig 3-1

4) Clock

This is to select an internal or external basic clock for the motor. The internal clock will use frequency generated inside the PPMC. In the external clock mode the clock is provided by the input on pin#39.

It is possible for PPMC101C to control speed from 50pps to 600pps with the internal clock 12.5KHz and for PPMC102A to control speed from 100pps to 1,200pps. To control speed from 400pps to 5Kpps a 100KHz external clock should be used for PPMC101C. The external clock must be lower than 133KHz. In case of PPMC102A, speed can be controlled from 800pps to 10Kpps with 200KHz external clock which must be lower than 244KHz.

5) Excitation Signal Switching Output

Usually the current into the stepper motor remains at maximum current during stanstill. This maximum current which holds the motor can cause overheating. Bit 5 is used to prevent this type of problem by switching '1' or'0'. PPMC can switch the excitation output to minimize the excitation current. Switching frequency is about 2.2KHz for PPMC101C with a duty cycle of 30% and 4KHz for PPMC102A with a duty cycle of 35%.

About 100 msec after the output of a phase excitation signal, the motor will start operating from a standstill when "switching" is selected.

Type of motor code, excitation method and logic level of excitation pulse output cannot be changed once they are set after RESET, while clock, excitation pulse switching output and initialization data can be changed.

3-2-2 Initialization Data

3-2-2-1 Pulse Rate

PPMC applies the idea of pulse rate (RA) to decide speed of the stepper motor. The relationship between pps and RA is expressed in the following equations :

$$pps = \frac{fo}{RA}$$

fo : Basic clock frequency

RA : Pulse Rate

pps : Motor pulse per second

Basic clock is either a 12.5KHz for PPMC101C (22.9KHz for PPMC102A) clock generated inside PPMC or external clock applied to pin#39. Bit 4 (clock command bit) in initialization command is used to select either the internal/external clock.

PPS Table 3-1 shows various logical figure of RA and practical use.

fo \ RA		Logical (2 – 255)	Practical (20 – 255)	
Internal clock	12.5 KHz	49 Hz – 6.25 KHz	49 Hz – 625 Hz	(PPMC101C)
	22.9 KHz	89 Hz – 11.4 KHz	89 Hz – 1,140 Hz	(PPMC102A)
External clock	100 KHz	392 Hz – 50 KHz	392 Hz – 5 KHz	(PPMC101C)
	200 KHz	784 Hz – 100 KHz	784 Hz – 10 KHz	(PPMC102A)

The appropriate number of pulse for acceleration/deceleration should be decided by the customer's experience, because it depends primarily on type of motor, inertia moment of load, etc. In case of large inertia moment of load, a large number of pulse for acceleration/deceleration should be selected for slow operation. PPMC can be adapted quite well to the majority of the load. With 2 byte to store the number of pulse needed for acceleration/deceleration, 4 - 8,160 steps can be set.

Some motors have a sympathetic point where there is no torque at certain frequency as shown in Fig 3-2. In such cases, the motor has to be started with a speed lower than the sympathetic point in order to fly into a higher speed area. To minimize the time to stay on the sympathetic point, higher speed for acceleration/deceleration must be applied. It is recommended that a damper should be used to increase the inertia moment if the motor goes into the sympathetic point with a small load.

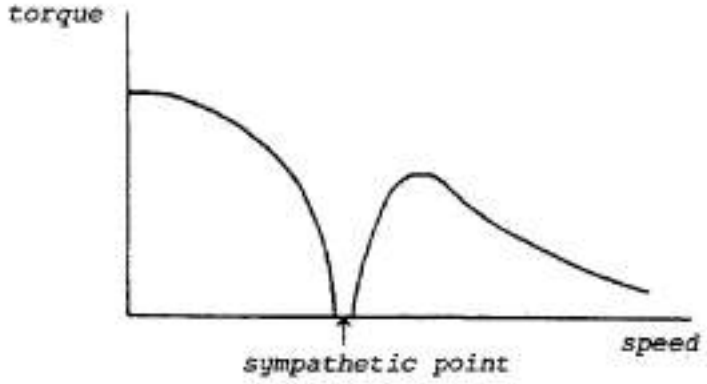


Fig 3-2

Initialization Data

Stepper motor has two types of operation as follows :

(A) Constant speed operation at lower speed of self-starting frequency.

(B) Ramp up/down operation.

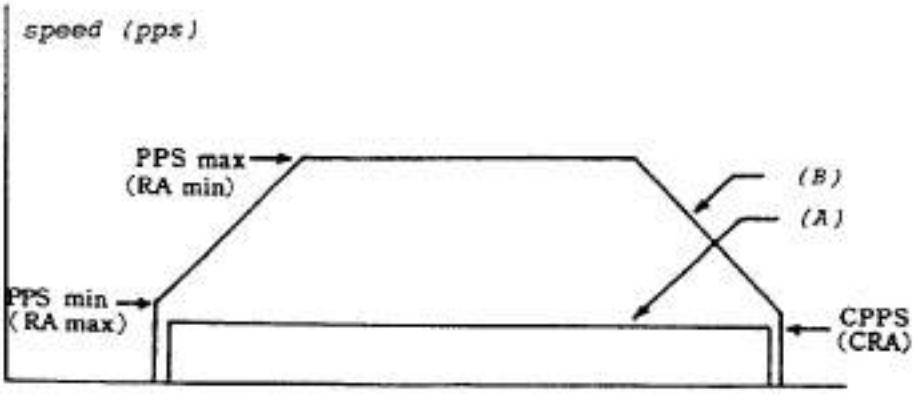


Fig 3-3

Self-starting frequency, parameters for high speed operation and acceleration/deceleration depend on the type of motor, excitation method, inertia or nature of load, etc. Relation among those are indicated below :

$$255 \geq CRA \geq RA\ max > RA\ min \geq 20$$

Pulse Rate	Symbol
At constant speed operation	CRA
At self-starting	RA max
At high speed operation	RA min

CRA and RA max are to be set as large as possible within the above limit. The hardware limits the external clock rate to a maximum of 133KHz for PPMC101C (244KHz for PPMC102A). For example with 100KHz external clock for PPMC101C and 200KHz for PPMC102A the above relation can be converted into pps (pulse per second) as following.

PPMC101C	$392Hz \leq CPPS \leq PPS\ min < PPS\ max \leq 5KHz$
PPMC102A	$784Hz \leq CPPS \leq PPS\ min < PPS\ max \leq 10KHz$

In conclusing, with a 100KHz external clock for PPMC101C or 200KHz for PPMC102A, the stepper motor can be controlled from 400pps to 5Kpps for PPMC101C or 800pps to 10Kpps for PPMC102A. For lower speed operation, external clock frequency should be slowed down accordingly.

3-2-2-3 Aberration of motor speed

There are two major sources that cause the motor speed to deviate from the theoretical value (see Fig 3-4). The first source of error derives from the execution time of routine that outputs the excitation. A 50 μsec overhead time is needed in addition to the delay timing for the pulse output. Therefore for slow speed operations, the 50 μsec error is insignificant. The % error of the output speed will increase with an increase in motor speed.

The second source of error is the non-synchronization of the basic clock and the internal timer. A randam error corresponding to ±1 basic clock pulse in the timer counter is possible. Note the percentage error will be larger at a slower clock rate. For example at a clock rate of 20Hz, the randam error is ±5%, which may be acceptable in practical application.

Following is the curve that shows the differnce between theoretical value and practical speed at a basic clock frequency of 100KHz (PPMC101C) and 200KHz (PPMC102A). The graph shows that at RA=15, the motor speed is 5Kpps for PPMC101C and 10KHz for PPMC102A.

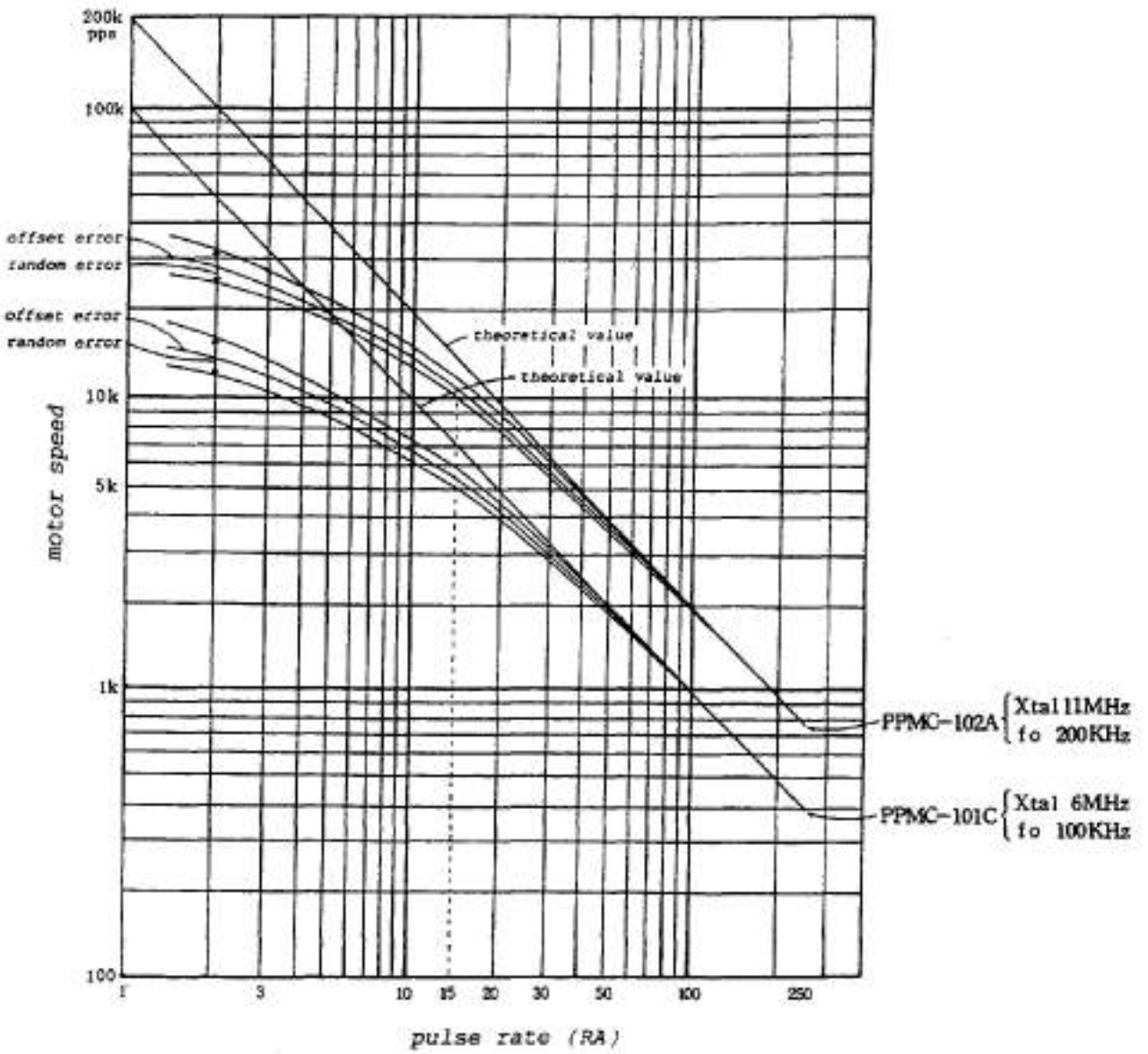
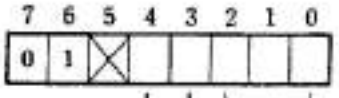


Fig 3-4

3-3 Operation Command



Operation command code

0 0 0 ..... Emergency stop
0 0 1 ..... Decelerating stop
0 1 0 ..... Single step
0 1 1 ..... Acceleration/Deceleration
1 0 0 ..... Constant speed operation
1 0 1 ..... To move until the limit
1 1 0 ..... To move until the high speed limit
1 1 1 ..... To move until the base point

Direction for rotation of motor

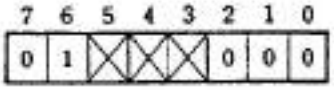
0 ..... C W
1 ..... C C W

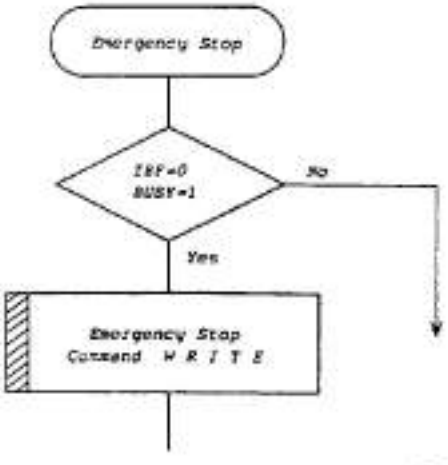
Interupt mask for FINISH signal

0 ..... Enable
1 ..... Disable

3-3-1 Emergency Stop

To stop rotation of motor instantaneously by inhibiting excitation output during any type of operation, whether it be acceleration/deceleration or constant speed operation. In high speed operation, the phase output stops instantaneously, but the motor will run off with inertia. Therefore, position data is no longer valid. It is necessary to reestablish the base point. During constant speed operation at self-starting frequency, motor can stop instantaneously and restarting is possible from that point by reading the number of operating pulse with the READ REGISTER COMMAND. Emergency stop requires only 1 byte operation command, and no data is necessary. Check whether the condition IBF=0, and BUSY=1 is satified before writing the emergency stop command as shown below.

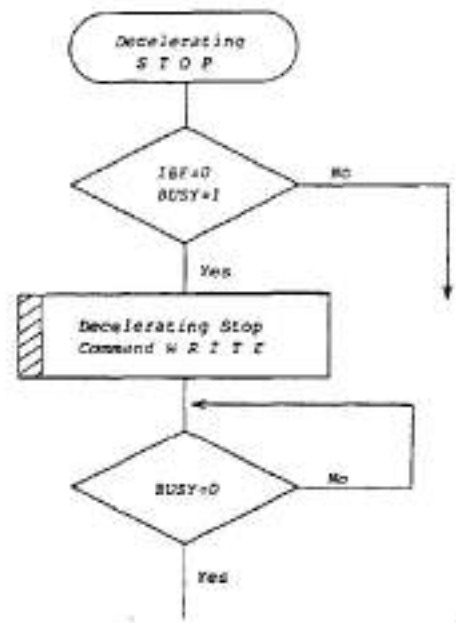




3-3-2 Decelerating Stop

When the decelerating stop command is input during acceleration/ deceleration, the motor will decelerate to stop. The motor will stop instantaneously during constant speed operation at self-starting frequency, the remaining number of pulse can be read, by the READ REGISTER COMMAND and the motor can be re-started from where it stopped. The bits for direction of motor rotation and FINISH INTERRUPT become assertive when the motor stops. Refer to the following flow chart for proper sequence of operation.

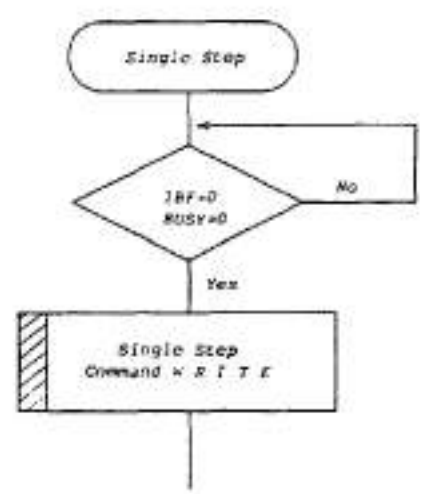
7	6	5	4	3	2	1	0
0	1	X	X	X	0	0	1



3-3-3 Single Step

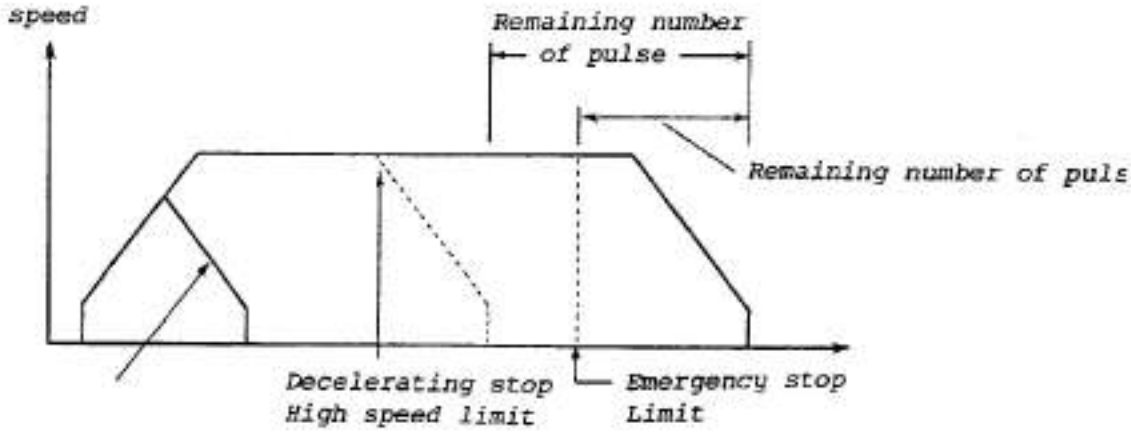
This is the command to move the stepper motor at a single step. It is useable when the master CPU needs to find out its position by itself. When this command is released continuously, timing must be controlled by the master CPU. All command modes are effective, this command consists of a single byte. No other data is necessary. Refer to the following flow chart for proper sequence of operation.

7	6	5	4	3	2	1	0
0	1	X			0	1	0



3-3-4 Acceleration/Deceleration

This command for acceleration/deceleration in accordance with the data at the time of initialization. In addition to the command itself this operation requires 3 bytes of data, which store the total number of pulses to be output. For triangle operation, total number of pulse must be smaller than two times the acceleration/deceleration pulse number. The limiting switch input L3, L4 can be used to trigger the deceleration (see Fig 2-2) and L1 and L2 can be used to stop the motor. Note that irrelevant signals from L1-L4 will be ignored. For example in Fig 2-2, if the carrier is moving CW, the signals from L2 and L4 will be ignored.



The number of pulse (step number requested for operation - 1) can be got with 3 bytes.
FFFFFF (Hexa decimal) input moves 16,777,216 steps which is the maximum number of steps to move at a time.

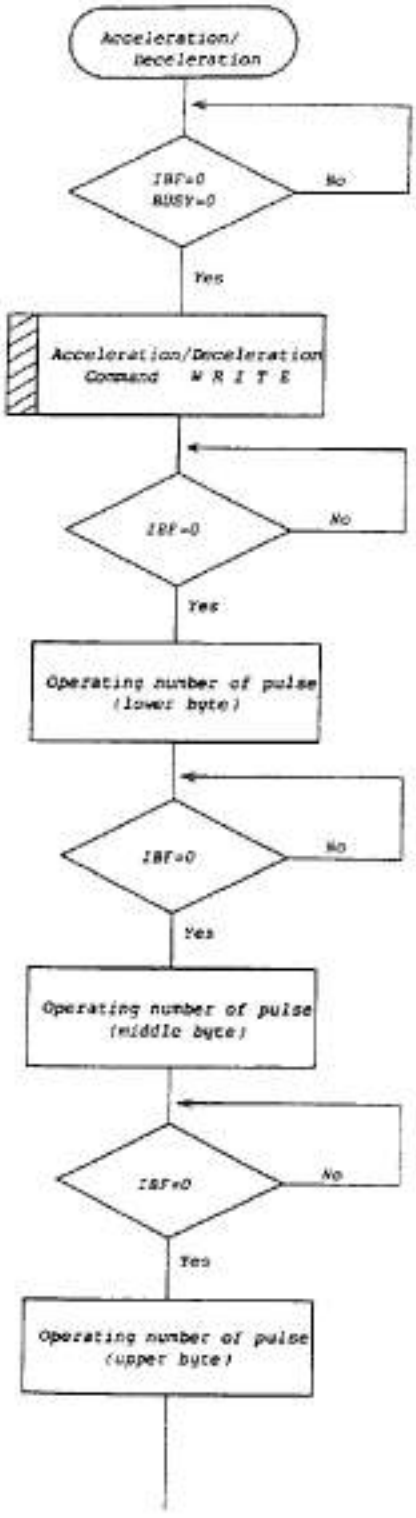
- Example -

To move 1,000 steps, 1,000 - 1 = 999 should be converted into Hexa decimal (0003E7) for input. Data must be input from the lower byte.

7	6	5	4	3	2	1	0
0	1	X			0	1	1

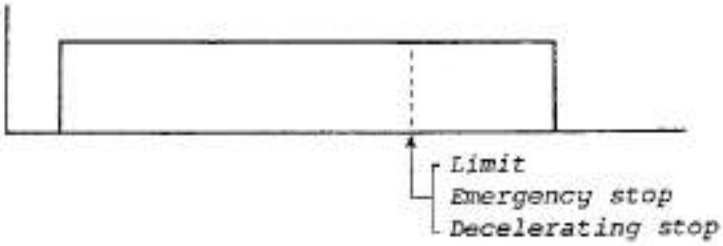
Operating number of pulse

	7	6	5	4	3	2	1	0	
(E7)	1	1	1	0	0	1	1	1	(lower byte)
(03)	0	0	0	0	0	0	1	1	(middle byte)
(00)	0	0	0	0	0	0	0	0	(upper byte)

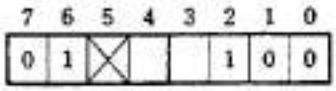


3-3-5 *Constant Speed Operation*

This command requires one byte of data for constant speed pulse rate as well as 3 bytes of data for the total number of pulse. The proper sequence of execution is shown in the following flow chart. The command causes the motor to rotate at a constant speed up to the designated distance. The speed is set by the pulse rate data, which has to be within the self-starting frequency of the motor. L1-L4 limit switch input can be used to decelerate and stop the motor. The READ REGISTER COMMAND can be used to readout the remaining number of pulse and the cause for stopping.



Refer to the following flow chart for proper sequence of operation.



Operating number of pulse

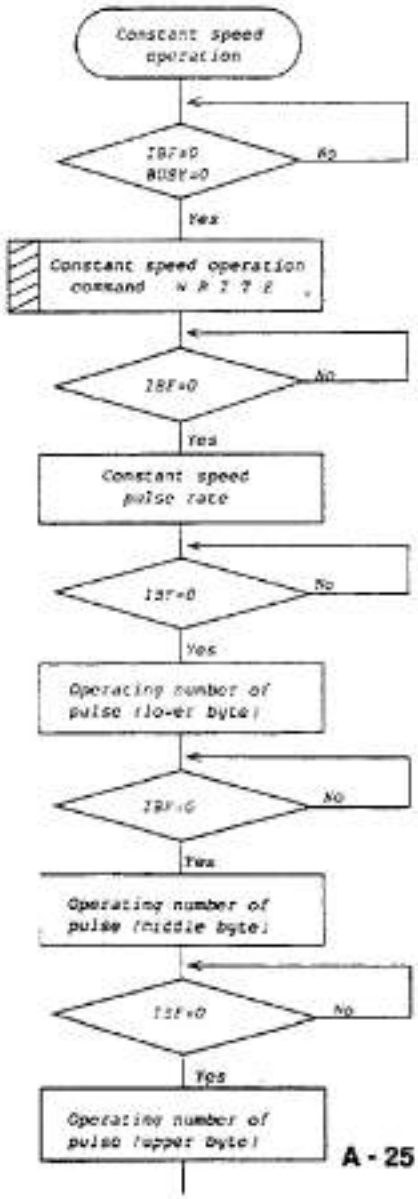


constant speed pulse rate

(lower byte)

(middle byte)

(upper byte)

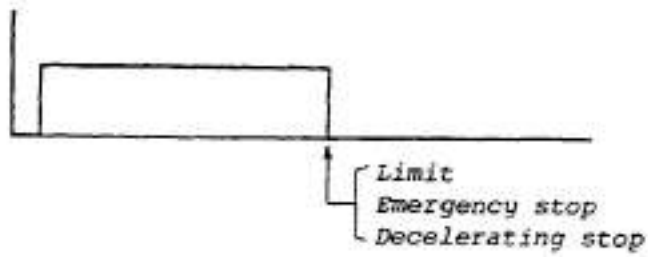


3-3-6 To Move At Constant Speed Until Limit Switch

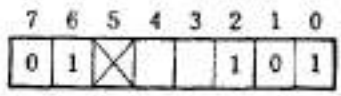
This mode of operation is similar to the previous one except that the data indicating the number of pulse is omitted. The motor will keep on moving until a signal from a limit switch (L1 or L2) is received. The signal from irrelevant L1 or L4 is ignored. Limit input in the same operating direction means as follows :

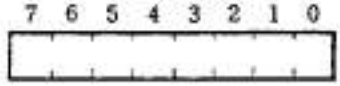
Operation	Input	
C W	L1	limit input
C C W	L2	" "

During a CW operation, L2 input will be ignored.

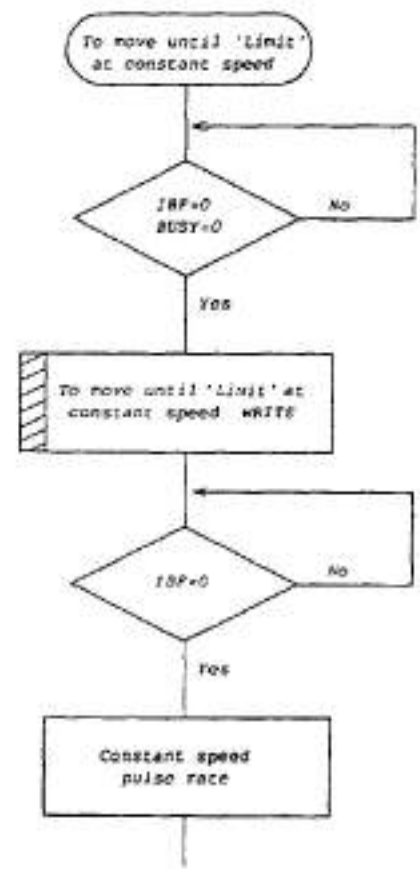


This command is normally used right after POWER ON or to re-start after motor run off.



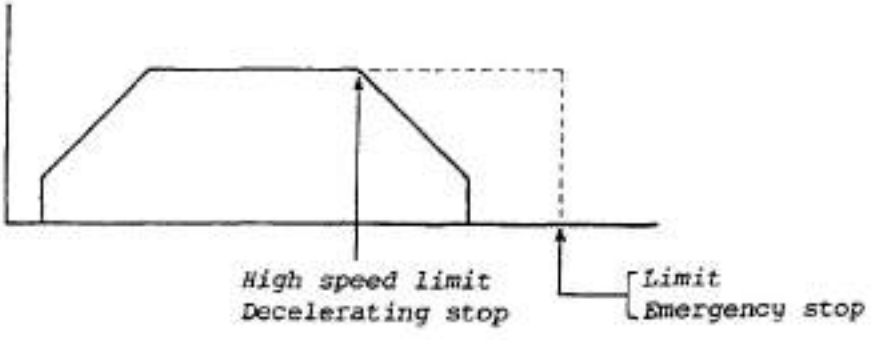


constant speed pulse rate



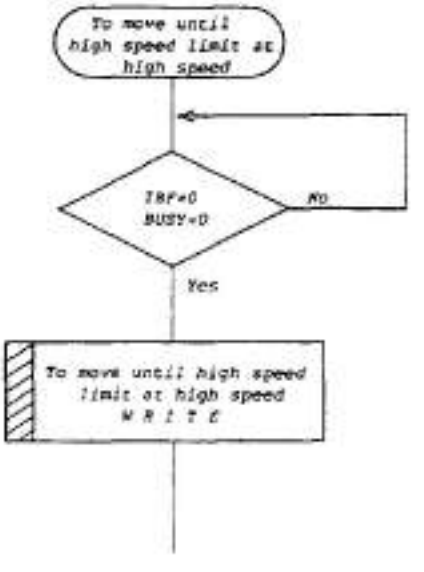
3-3-7 *To Move At High Speed Until High Speed Operation Limit*

This is the command to accelerate for high speed operation in accordance with the data at the time of initialization. Under this command the motor will rotate at high speed until a limiting signal is received. Then it will decelerate to stop according to the number of decelerating pulse. Once deceleration begins, it continues to decelerate even if limit input turns out '1'. Limit switch L1 for CW and L2 for CCW rotation can force the motor to stop, but inertia may cause the motor to overrun the desired stopping point.



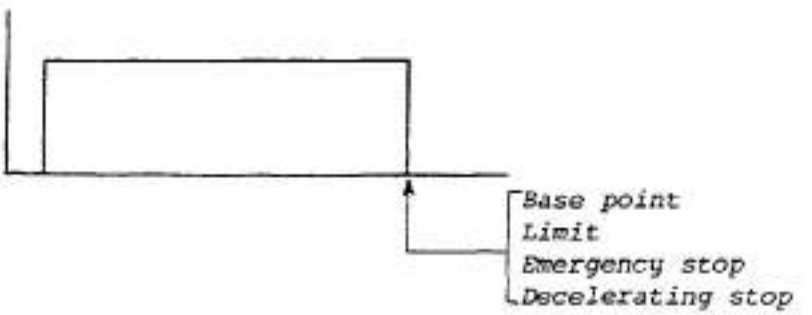
This command is also used right after POWER ON or to re-start motor after run-off. The decision whether to use this command or the command 'to move at constant speed until limit switch' (3-3-6) depends on the distance, time, accuracy of position, etc.

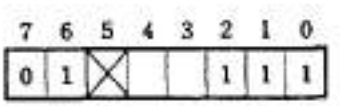
7	6	5	4	3	2	1	0
0	1	X			1	1	0

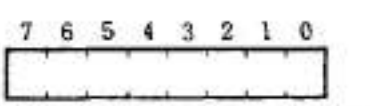


3-3-8 To Move To Base Point

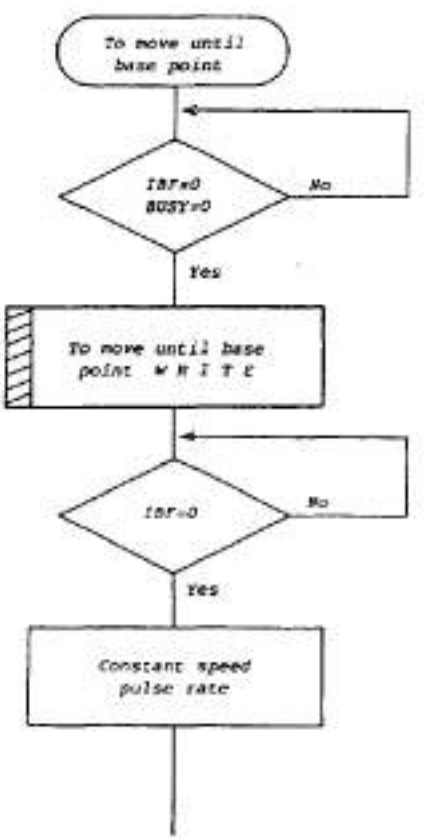
In this mode of operation, the motor rotates at constant speed until a $\overline{CNP}$ signal is detected. The motor can also be stopped by the relevant limit input L1-L4, emergency stop or decelerating stop.







constant speed pulse rate



3-4 READ REGISTER

READ REGISTER COMMAND is used to read three kinds of status and a 3 byte data during standstill of motor. The proper format of the command is indicated below :

7	6	5	4	3	2	1	0
1	0	0	0	0	0		

Register code

0 0 ..... FINISH status

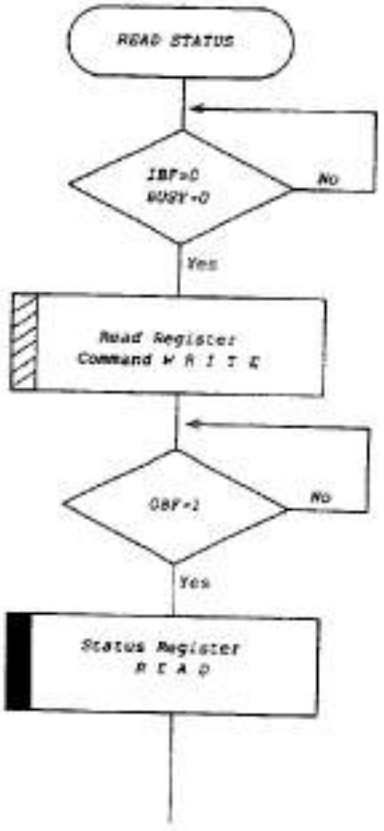
0 1 ..... Input signal status

1 0 ..... Output signal status

1 1 ..... Remaining number of pulse

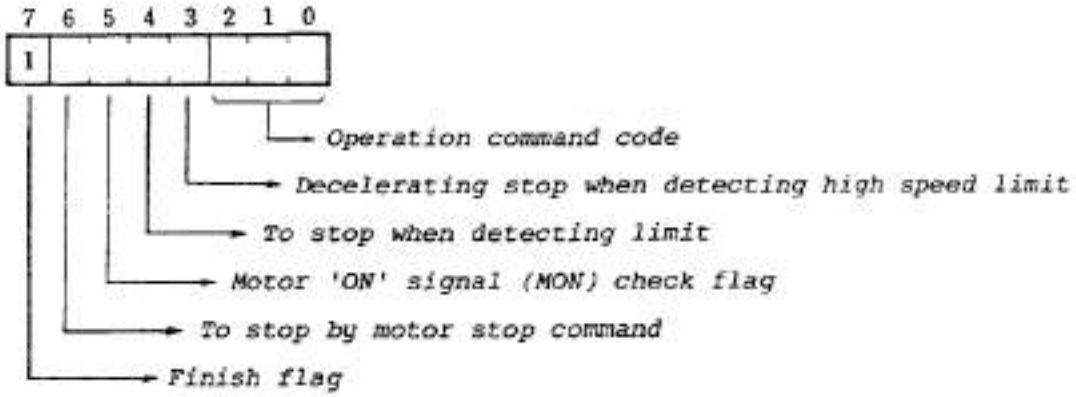
3-4-1 READ FINISH STATUS

The proper sequence to read the various status is shown in the following flow chart.



3-4-1-1 FINISH STATUS

The finish status register contains the following information :



The lower 3 bytes contain the operation command code.

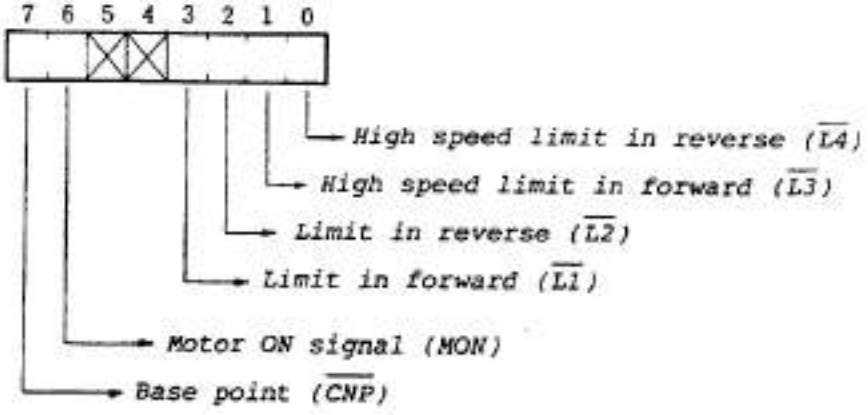
Bit 3 or 4 goes up to '1' after high speed limit ($\overline{L3}$, $\overline{L4}$) or limit ($\overline{L1}$, $\overline{L2}$) is input for motor stop. Bit 5 shows '1' when motor cannot operate with 'motor on signal' (MON) = 0.

A '1' in bit 6 indicates either an emergency stop or decelerating stop. When all number of pulses for acceleration/deceleration and constant speed operation are completely consumed to stop, all bits from 3 to 6 turns to '0'.

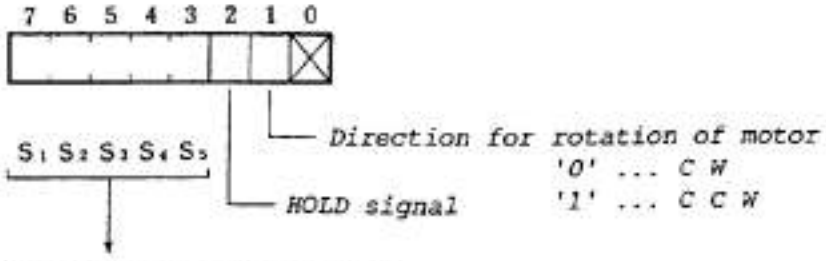
In the absence of finish interrupt mask, $\overline{INT}$ becomes assertive at the end of the operation, $\overline{INT}$ signal can be cleared to '1' by reading finish status and released.

3-4-1-2 INPUT signal

The input signal register reflects the state of various inputs shown below at the point where the motor stops.



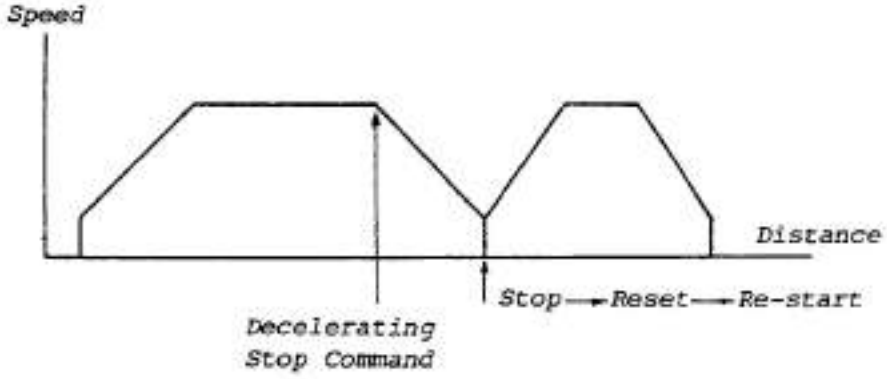
3-4-1-3 OUTPUT Signal



Phase outputs for stepper motor can be checked by bit 3 to 7.

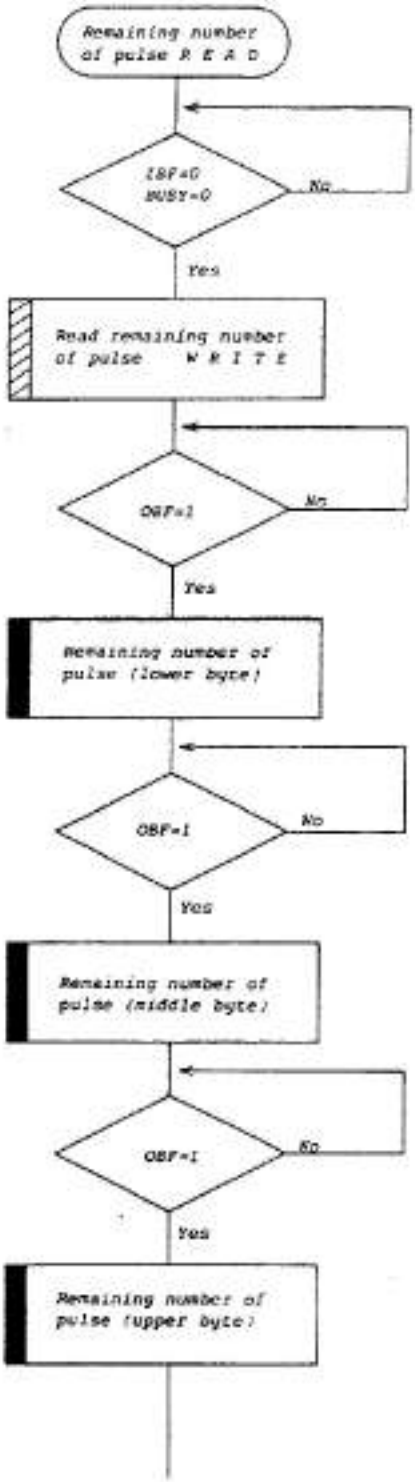
3-4-2 Remaining number of pulse

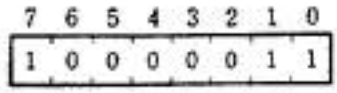
When the motor is stopped by limit switch or stop command during acceleration/deceleration or constant speed operation, the remaining number of pulse can be read by using this mode. If it is decided to finish the operation after the stop command, the original command and the remaining number of pulse can be input again to restart the mode.



The data becomes '0' when the operation has been successfully terminated.

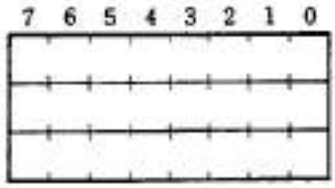
The following flow chart shows the proper sequence of programming in order to read the data from PPMC.





Command

Remaining number of pulse



(lower byte)

(middle byte)

(upper byte)

APPENDIX **B**

SOURCES FOR STEPPER MOTORS & ASSEMBLIES

B.1 GENERAL INFORMATION

MetraByte stocks a standard stepper motor and incremental shaft encoder (models STEP-MOT1 & STEP-ENC1). Their specifications are provided in Appendix C. Although these have been selected from very popular types, they may not be suitable for your application. This Appendix details vendors who specialize in stepper motors and associated mechanical components.

In the course of assembling a stepper motor system, you may need to procure stepper motors, lead screws, X-Y tables, shaft encoders etc. Many vendors provide this type of equipment and the following list has been compiled to assist you in locating components. Inclusion of a vendor in this list does not imply that MetraByte Corporation endorses the vendor or is in any way responsible for the performance of its equipment. Likewise the list is not complete, and omission of any vendor is purely coincidental. This list is provided purely for your convenience and is not exhaustive.

B.2 STEPPER MOTORS

1. Superior Electric Company
   383 Middle Street
   Bristol CT 06010 Phone: 203-582-9561
2. B & B Motor & Control Corp.
   Apple Hill Commons
   Burlington CT. 06013 Phone: 203-673-7151
3. Bodine Electric Company
   2500 W. Bradley Place
   Chicago IL. 60618 Phone: 312-478-3515
4. Litton Clifton Precision
   P.O. Box 160
   Murphy N.C. 28906-0160 Phone: 704-837-5115
5. Oriental Motor U.S.A. Corp.
   2701 Plaza Del Amo Suite 702
   Torrance, CA 90503 Phone: 213-515-2264

B.3 SHAFT ENCODERS

1. Motion Control Devices
   80 Stedman Street
   Lowell MA. 01851 Phone: 617-454-3407

2. B.E.I. Motion Systems Company
   Computer Products Division
   1755-B La Costa Meadows Drive
   San Marcos CA. 92069 Phone: 619-471-2600

3. Datametrics
   Dresser Industries Inc.
   340 Fordham Road
   Wilmington MA. 01887 Phone: 617-658-5410

4. Honeywell Disc Instruments Inc.
   102 East Baker Street
   Department TR
   Costa Mesa CA. 92626 Phone: 714-979-5300

B.4 X-Y TABLES, SLIDES, LEADSCREWS ETC.

1. Daedal Inc.
   P.O. Box G
   Sandy Hill Road
   Harrison City PA. 15636 Phone: 1-800-245-6903
   (in PA) 412-744-4451

2. New England Affiliated Technologies Inc.
   620 Essex Street
   Lawrence MA. 01841 Phone: 617-685-4900

3. B & B Motor & Control Corp.
   Apple Hill Commons
   Burlington CT. 06013 Phone: 203-673-7151

4. Klinger Scientific Corporation
   110-120 Jamaica Avenue
   Richmond Hill NY. 11418 Phone: 718-846-3700

B.5 TRANSLATORS, STEPPER DRIVERS

1. Superior Electric Company
   383 Middle Street
   Bristol CT. 06010 Phone: 203-582-9561

2. Bodine Electric Company
   2500 W. Bradley Place
   Chicago IL. 60618 Phone: 312-478-3515

3. Oriental Motor U.S.A. Corp.
   369 Passaic Avenue
   Fairfield N.J. 07006 Phone: 201-882-0480

B.6 TOOTHED BELTS, GEARS, DRIVE COMPONENTS ETC.

1. Precision Industrial Components Corporation
   P.O. Box 1004
   Benson Road
   Middlebury CT. 06762 Phone: 1-800-243-6125
   in CT: 203-758-8272

B.7 OTHER SOURCES

If you are on a tight budget, Herbach & Rademan usually carry a range of new and used stepper motors and other mechanical odds and ends. Ask for their catalog, their address is

Herbach & Rademan Corporation
401 E. Erie Avenue
Philadelphia PA. 19134 Phone: 215-426-1708

* * * * *

APPENDIX **C**

STEP-MOT1 & STEP-ENC1 SPECIFICATIONS

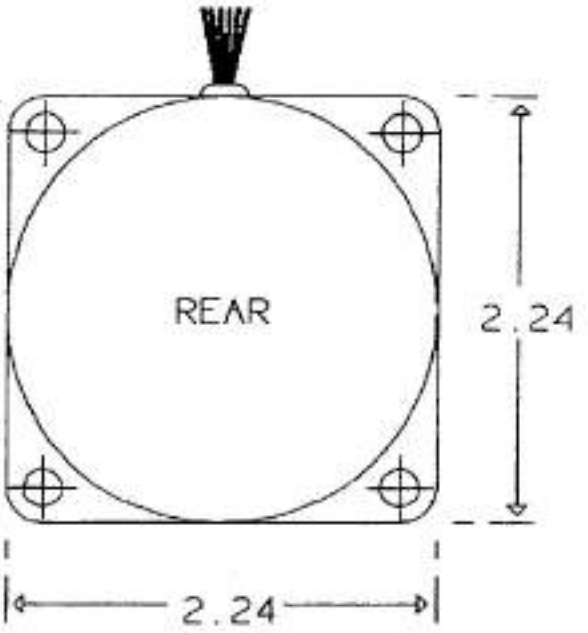
C.1 STEP-MOT1 SPECIFICATION

Electrical Specifications

Nominal DC Volts	5.0 volts
Winding Resistance	5.0 ohms at 25 deg. C.
Rated Current	1.0 amps per winding
Winding Inductance	10.4 millihenries
Winding Type	4-phase, 6-lead unipolar
Time for Single Step	2.5ms with 24VDC drive

Mechanical Specifications

Step angle	1.8 degrees full step.
Angle Accuracy	5%
Holding Torque	53 oz.-in. min. (2 windings energized)
Residual Torque	1.25 ox.-in. min.
Rotor Inertia	0.04 lb.-in^2
Torque/Inertia Ratio	32,000 typ.
Shaft Diameter	0.25 inch
Radial Load	15 lbs. max.
Axial Load	25 lbs. max.
Weight 20 oz.	



SPECIFICATIONS

RATING: 5v 1 amp 5 ohm winding

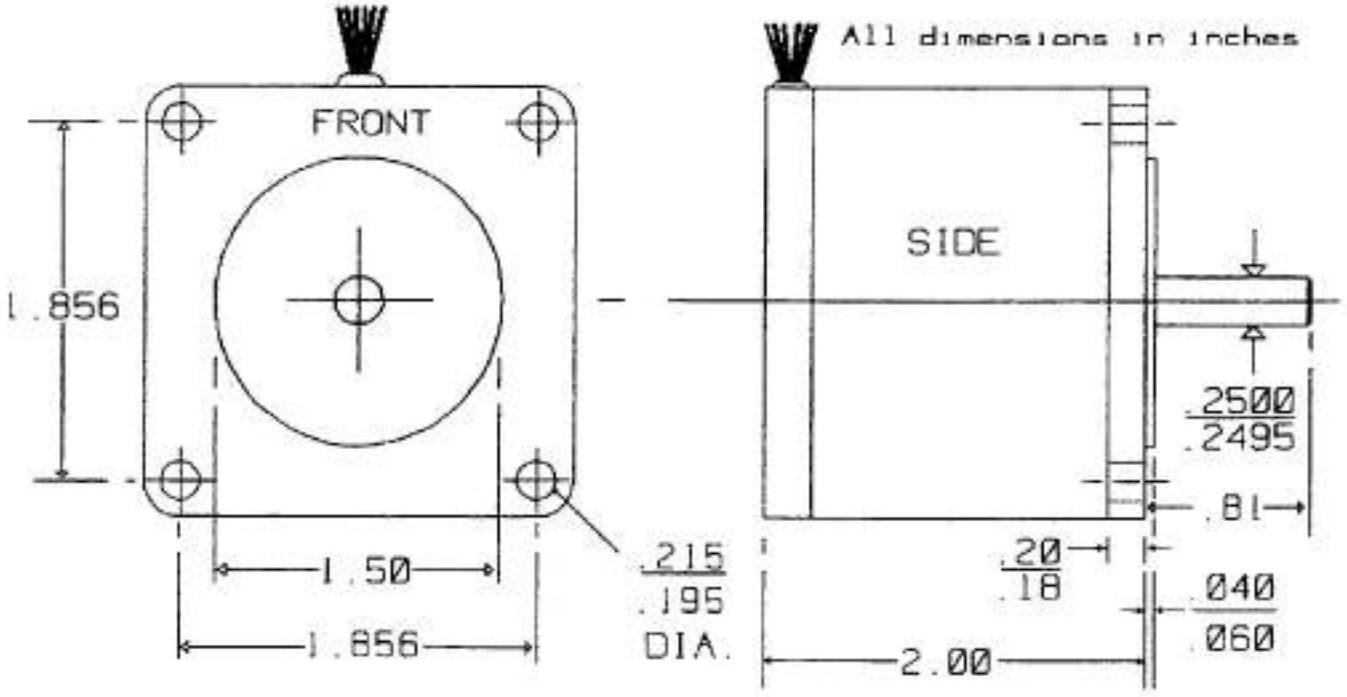
STEPS: 200 per rev. (1.8 deg.)

TORQUE: 53 oz-in holding

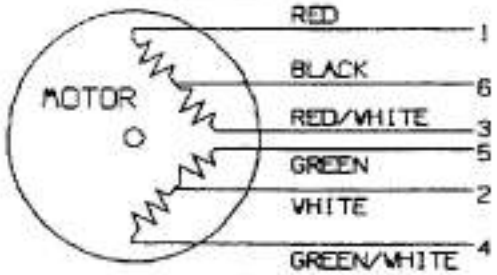
WEIGHT: 20 oz.

INDUCTANCE: 10.4 mH

All dimensions in inches



CONNECTIONS



STEP-MOTI SPECIFICATION

C.2 STEP-ENC1 SPECIFICATION

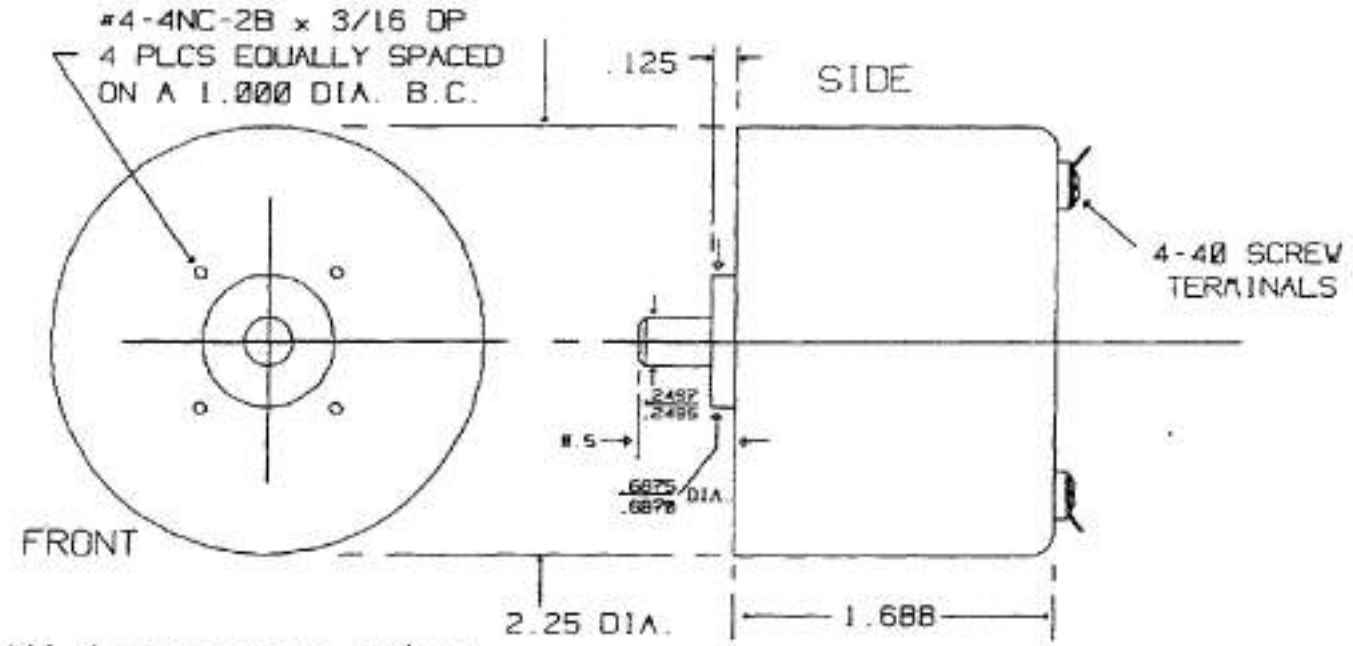
Electrical Specifications

Output Channels	Two count channels electrically phased 90 degrees +/-20 degrees. One zero reference (index pulse) signal channel, half count per revolution.
Count Channel Accuracy	Pulse to any other pulse, 1 minute of arc Pulse to adjacent pulse, 30 seconds of arc
Count Channel Signal	Square wave, TTL compatible. Logic 0, 0.4v max. at 16mA sink current Logic 1, 2.4v min. at 800uA source current
Symmetry (Duty Cycle)	180 degrees +/-10 degrees
Frequency Response	0 to 50KHz max.
Rise & Fall Time	Less than 200 nanoseconds
Operating Temperature	0 to 70 degrees C.
Power Consumption	5v +/-5% @@ 155mA
Terminal Connections	

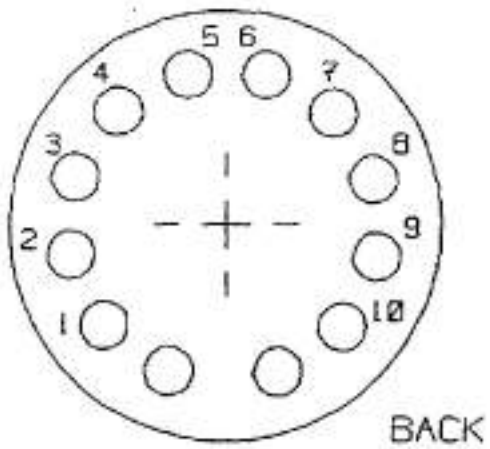
TERMINAL NUMBER	FUNCTION
1	+5v
2	GND.
4	Channel A
5	Channel B
6	Index pulse

Mechanical Specifications

Shaft Speed	3000 r.p.m. at max. shaft loading 5000 r.p.m. at lower loads
Shaft Loading	Radial, 3lbs. max. at end of shaft Axial, 3lbs. max.
Starting Torque	0.01 oz.-in.
Rotary Inertia	0.0002 oz.-in.-sec@+[2]
Maximum Acceleration	50,000 rad/sec^2
Materials	Shaft type 303 stainless steel Mounting boss aluminum Housing polycarbonate resin
Bearings	Instrument class, shielded
Weight 4 oz.	



All dimensions in inches



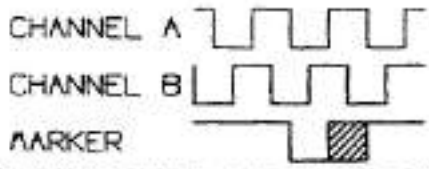
SPECIFICATIONS

RESOLUTION: 1000 LINES/REV
MAX. SPEED: 3000 RPM
STARTING TORQUE: 0.01 oz-in
INERTIA: 0.0002 oz-in-sec2
MAX. ACCELERATION: 50,000 RAD/SEC2
SHAFT LOADING: RADIAL 3 lbs max.
AXIAL 3 lbs max.

POWER: +5v at 155mA
CHANNELS: A & B AND MARKER
OUTPUTS: TTL LEVELS . 16mA SINK

CONNECTIONS

1 - +5v power
2 - COMMON (GND)
4 - A OUTPUT
5 - B OUTPUT
6 - MARKER OUTPUT



CHANNEL B LEADS CHANNEL A FOR CLOCKWISE ROTATION

STEP-ENC1

SPECIFICATION

APPENDIX **D**

MSTEP-5 & STA-STEP SPECIFICATIONS

D.1 POWER CONSUMPTION

+5v supply (MSTEP-5 only)	1.7A typ., 2A max.
+5v supply (with STA-STEP)	1.8A typ., 2.1A max.
+12v supply	not used
-12v supply	not used
-5v supply	not used

D.2 MSTEP-5 SPECIFICATIONS

Stepper channels	2 (individually programmable)
Maximum step count	+/-16,777,215
Maximum step rate	5000 p.p.s.
Acceleration/deceleration ramping	automatic trapezoidal programmable start, run & ramping rates.
Limit switch inputs (active low, open collector, TTL or mechanical switch to ground)	5 per channel (end of travel, high speed & base point)
Translator Drive	CCW/CW and negative going 5 microsecond pulse.
Phase (winding) drives	TTL compatible signals for 3,4 or 5 phase motor windings.
Phase drive logic polarity	programmable
Phase drive sequence	programmable full or half step.
Power chopping at standstill	programmable, 2.2 KHz 30% duty cycle (reduces motor heating)

Encoder channels	2
Type	Incremental quadrature
Maximum encoder count	+/-8,388,607
Input	TTL/CMOS compatible with 2 stage digital noise filter.
Maximum encoder pulse rate	25 KHz.

D.3 STA-STEP SPECIFICATIONS

Maximum motor voltage	24VDC
Maximum winding current	5A/phase
Motor circuit isolation from computer ground	350v max.
Driver type	L/R
Maximum step rate	2000 p.p.s.
Driver transistor on resistance	0.08 ohm

D.4 LOGIC OUTPUTS

All outputs	TTL compatible 0.4v max output low voltage 2.4v min output high voltage
Phase drive, S1-S5, PULSE, CW/CCW and HOLD sink current.	24mA at 0.5v
Phase drive, S1-S5, PULSE, CW/CCW and HOLD source current	-3mA at 2.4v
Encoder carry/borrow output sink current.	8mA at 0.5v
Encoder carry/borrow output source current.	-0.4mA at 2.4v

D.5 LOGIC INPUTS

All inputs	TTL/CMOS compatible 0.8v max input low voltage 2.0v min input high voltage
Pullups	All inputs have internal 10K pullups to +5v. (5K with STA-STEP attached)

D.6 POWER OUTPUTS

IBM P.C. buss supplies	+5v, +12v & -12v
Loading	Dependent on other peripherals & computer type, see Note.

NOTE: Due to connector and cable ratings, it is recommended that +5v power be limited to 2A, and that +/-12v power to the lesser of 1A or whatever is available. Excitation of the stepper motor from the computer is **not recommended** .

D.7 PHYSICAL & ENVIRONMENTAL SPECIFICATIONS

Operating temperature range	0 to 50 deg.C.
Storage temperature range	-20 to 70 deg.C.
Humidity	95% non-condensing
Weight	8oz. (230 gm.)

D.8 CONNECTOR PINOUT

Rear view of 50 pin connector
(O) = output, (I) = input

CH. A Limit 1, L1 (I)	1	2	CH. A Limit 2, L2 (I)
CH. A Limit 3, L3 (I)	3	4	CH. A Limit 4, L4 (I)
CH. A Base point, CNP (I)	5	6	CH. A Motor control (I)
CH. A direction CCW/CW (O)	7	8	GND.
CH. A S1 phase output (O)	9	10	CH. A S2, phase output (O)
CH. A S3, phase output (O)	11	12	CH. A S4, phase output (O)
CH. A S5, phase output (O)	13	14	CH. A PULSE OUT (O)
CH. A HOLD ACK. (O)	15	16	GND.
GND.	17	18	CH. B Limit 1, L1 (I)
CH. B Limit 2, L2 (I)	19	20	CH. B Limit 3, L3 (I)
CH. B Limit 4, L4 (I)	21	22	CH. B Base point, CNP (I)
CH. B Motor control (I)	23	24	CH. B Direction CCW/CW (O)
GND.	25	26	CH. B S1, phase output (O)
CH. B S2, phase output (O)	27	28	CH. B S3, phase output (O)
CH. B S4, phase output (O)	29	30	CH. B S5, phase output (O)
CH. B PULSE OUT (O)	31	32	CH. B HOLD ACK. (O)
GND.	33	34	GND.
CH. A Encoder A input (I)	35	36	CH. A Encoder B input (I)
CH. A counter out (O)	37	38	CH. B Encoder A input (I)
CH. B Encoder B input (I)	39	40	CH. B counter out (O)
CH. A external clock (I)	41	42	CH. B external clock (I)
+12v	43	44	no connection
Power from -12v	45	46	no connection
computer +5v	47	48	+5v
GND.	49	50	GND.

Keying blocks are present between pins 3 & 5 and 47 & 49. Connector attached to board is 3M (Scotchflex) # 3433-5303. Mating half for ribbon (insulation displacement cable) is 3M # 3425-6050

D.8 SIGNAL DESCRIPTIONS

Phase outputs S1-S5	TTL outputs that provide signals for motor winding excitation. Logic polarity and step sequence (full/half) are programmable.
CCW/CW	Direction signal output 0 = clockwise 1 = counterclockwise.
PULSE OUT	Pulse output corresponding to steps. 5 microsecond negative going pulses. CCW/CW and PULSE can be used to drive a translator.
HOLD ACK.	If switching at standstill is enabled, HOLD ACK. goes high 3 milliseconds after motor stops otherwise it is always low.
L1 & L2 limit inputs	Active low overtravel limit inputs. Motor will stop immediately on encountering either of these limits. L1 - clockwise limit L2 - counterclockwise limit

L3 & L4 limit inputs	Active low high speed limit inputs. Motor will perform a decelerating stop at either of these limits if executing an accelerate or proceed to high speed limit command. L3 - clockwise limit L4 - counterclockwise limit
Base point, CNP input	Active low home or reference point limit switch input. Motor will stop at base point if executing a proceed to base point command.
Motor control input	Motor control input must be high for controller to execute commands. It can be used to monitor power on the stepper motor.
Encoder A & B inputs	Correspond to the A & B quadrature outputs of the incremental shaft encoder.
Counter out	Carry/borrow overflow bit from the 24 bit internal encoder counter.

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP