
**UNIDEX[®] 14
OPERATORS
MANUAL**

PN: EDU 117



**AEROTECH, INC., 101 Zeta Drive, Pittsburgh, PA 15238
(412) 963-7470 • TWX 710-795-3125 • FAX(412)963-7459**

Aerotech Inc.
Release No. 2.0
November, 1991

DISCLAIMER:

The information contained in this manual is subject to change due to improvements in design. Though this document has been checked for inaccuracies, Aerotech does not assume responsibility for any errors contained herein.

TRADEMARKS:

Unidex is a registered trademark of Aerotech, Inc.

IBM PC/XT/AT are registered trademarks of the International Business Machines Corporation.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

SECTION 1-1: GENERAL DESCRIPTION	1-1
SECTION 1-2: SPECIFICATIONS	1-3
1-2-1: FUNCTIONAL SPECIFICATIONS	1-3
1-2-2: UNIDEX 14 BLOCK DIAGRAM	1-5

CHAPTER 2: GETTING STARTED

SECTION 2-1: INTRODUCTION	2-1
SECTION 2-2: THEORY OF OPERATION	2-2
SECTION 2-3: U14C	2-3
2-3-1: U14C JUMPER CONFIGURATIONS	2-3
2-3-2: HARDWARE INSTALLATION	2-8
2-3-3: SOFTWARE INSTALLATION	2-9
2-3-4: MOTOR CONTROL CONNECTOR	2-10
SECTION 2-4: UNIDEX 14 INTERFACE BOARD	2-12
2-4-1: JUMPER CONFIGURATIONS	2-12
2-4-1-1: INPUT VOLTAGE	2-12
2-4-1-2: OPTIONAL ENCODER FEEDBACK	2-12
2-4-1-3: INPUT/OUTPUT BITS OR SYSTEM CONTROL	2-13
2-4-1-4: AXIS AUXILIARY OUTPUTS	2-14
2-4-1-5: LO/HI CURRENT CONTROL OR CZ SELECT	2-15
SECTION 2-5: VELOCITY PROFILES	2-17
2-5-1: FUNCTIONAL DESCRIPTION	2-17
2-5-2: VELOCITY PROFILE OPTIONS	2-18
2-5-2-1: LINEAR RAMPS	2-18
2-5-2-2: PARABOLIC RAMPS	2-19
2-5-2-3: COSINE RAMPS	2-20

CHAPTER 3: I/O CHANNEL INTERFACE

SECTION 3-1: I/O CHANNEL	3-1
3-1-1: PC/XT/AT DATA BUS	3-1
3-1-2: PC/XT/AT ADDRESS BUS	3-1
3-1-3: MEMORY AND I/O CONTROL	3-1
3-1-4: INTERRUPT REQUEST	3-2
3-1-5: DIRECT MEMORY ACCESS	3-2
3-1-6: CLOCK AND OSC LINES	3-3
3-1-7: RESET DRV	3-3
3-1-8: I/O CH RDY	3-3

SECTION 3-2: ADDRESS SELECTION	3-3
SECTION 3-3: USING INTERRUPTS	3-5
SECTION 3-4: DMA	3-5
SECTION 3-5: I/O REGISTERS	3-6
3-5-1: DATA REGISTER	3-6
3-5-2: DONE FLAG REGISTER	3-6
3-5-3: INTERRUPT AND DMA CONTROL REGISTER	3-7
3-5-4: STATUS REGISTER	3-7
SECTION 3-6: POWER SUPPLY REQUIREMENTS	3-8

CHAPTER 4: CHASSIS, I/O AND DRIVER INTERFACE

SECTION 4-1: INTRODUCTION	4-1
SECTION 4-2: UNIDEX 14 REAR PANEL CONECTIONS	4-2
4-2-1: UNIDEX 14 CONTROL CONNECTIONS	4-5
4-2-2: LIMIT SWITCH, MARKER AND ENCODER	4-7
4-2-2-1: CONNECTOR SPECIFICATIONS	4-7
4-2-2-2: SPECIFICATION	4-12
SECTION 4-3: AUXILIARY CONTROL CONNECTIONS	4-16
4-3-1: CONNECTOR SPECIFICATIONS	4-16
4-3-2: CONNECTOR SPECIFICATIONS	4-17
4-3-3: AUXILIARY CONTROL ELECTRICAL SPECIFICATIONS	4-18
SECTION 4-4: OPTO-ISOLATED I/O INTERFACE CONNECTOR	4-19
4-4-1: J2 CONNECTOR SPECIFICATIONS	4-20
SECTION 4-5: SERIAL OUTPUT INTERFACE	4-22
SECTION 4-6: UNIDEX 14 POWER CONNECTIONS	4-29

CHAPTER 5: INTERNAL STRUCTURE OF UNIDEX 14

SECTION 5-1: DESCRIPTION OF THE DRIVE CHASSIS	5-1
SECTION 5-2: DESCRIPTION OF THE U14C CONTROL BOARD	5-7

CHAPTER 6: DRIVE MODULES

SECTION 6-1: STEPPING DRIVES	6-1
6-1-1: DM4001 AND DM4005 STEPPING DRIVES	6-1
6-1-1-1: CIRCUIT DESCRIPTION	6-1
6-1-1-2: "HOME" REFERENCE DEFINITION AND OPTIONS	6-2
6-1-1-3: OUTLINE OF MARKER BUFFER CIRCUIT	6-4
6-1-1-4: LIMIT SWITCH POLARITY SELECTION	6-5
6-1-1-5: +/- DIRECTION DEFINITION AND OPTIONS	6-5
6-1-1-6: SELECTING STEPPING MOTOR RESOLUTION	6-6
6-1-1-7: PERSONALITY MODULE	6-7
6-1-1-8: RCN1:8-9, HOME CLOCK OSCILLATOR ADJUSTMENT	6-8
6-1-1-9: RCN1: 3-14 AND RCN1: 2-15, LO/HI CURRENT LEVELS	6-8

6-1-2: DMV8008 AD DMV16008 DRIVE MODULES	6-14
6-1-2-1: CIRCUIT DESCRIPTION	6-14
6-1-2-2: "HOME" REFERENCE DEFINITION AND OPTIONS	6-14
6-1-2-3: OUTLINE OF MARKER BUFFER CIRCUIT.....	6-16
6-1-2-4: LIMIT SWITCH POLARITY SELECTION.....	6-17
6-1-2-5: +/- DIRECTION DEFINITION AND OPTIONS.....	6-18
6-1-2-6: SELECTING STEPPING MOTOR RESOLUTION.....	6-18
6-1-2-7: PERSONALITY MODULE	6-20
6-1-2-8: RCN1:8-9, HOME CLOCK OSCILLATOR ADJUSTMENT	6-21
6-1-2-9: RCN1: 3-14 AND RCN1: 2-15, LO/HI CURRENT LEVELS	6-21
SECTION 6-2: DSL3015,4020,8020 AND 16020DC SERVO DRIVE MODULES.....	6-26
6-2-1: CIRCUIT DESCRIPTION	6-26
6-2-2: DSL DC SERVO DRIVE MODULES OPERATION	6-26
6-2-3: OPTIONAL TACHOMETER	6-29
6-2-4: MOTOR/ENCODER/TACHOMETER PHASING	6-29
6-2-5: MOTOR LOAD FUSE RATINGS FOR FUSE F1/CUR LIM.....	6-29
6-2-6: "HOME" REFERENCE DEFINITION AND OPTIONS	6-30
6-2-7: MARKER INPUT CIRCUIT	6-32
6-2-8: LIMIT SWITCH POLARITY SELECTION.....	6-33
6-2-9: ENCODER MULTIPLICATION PARAMETERS.....	6-33
6-2-10: ADJUSTMENTS.....	6-34
6-2-11: ADJUSTING POSITION AND VELOCITY LOOP	6-37
SECTION 6-3: DAV4008 AND 16008 STEPPING DRIVE MODULES.....	6-48
6-3-1: CIRCUIT OPERATION	6-48
6-3-2: DAV DRIVE MODULE OPERATION.....	6-48
6-3-2-1: "HOME" REFERENCE DEFINITION AND OPTIONS	6-50
6-3-2-2: MARKER INPUT CIRCUIT	6-52
6-3-2-3: LIMIT SWITCH POLARITY SELECTION.....	6-53
6-3-2-4: ENCODER MULTIPLICATION PARAMETERS.....	6-53
6-3-2-5: ADJUSTMENTS.....	6-54
6-3-2-6: ADJUSTING POSITION AND VELOCITY LOOP	6-57
SECTION 6-4: POWER SUPPLY BOARD	6-64
6-4-1: CIRCUIT DESCRIPTION	6-64

CHAPTER 7: STEPPING DC SERVO AND AERODRIVE MOTORS

SECTION 7-1: HARDWARE SPECIFICATIONS	7-1
--	-----

CHAPTER 8: COMMAND STRUCTURE

SECTION 8-1: INTRODUCTION.....	8-1
SECTION 8-2: AXIS SPECIFICATION COMMANDS	8-3
SECTION 8-3: SYSTEM CONTROL COMMANDS.....	8-6
SECTION 8-4: COMMANDS FOR USER I/O	8-12
SECTION 8-5: MOVE SPECIFICATION COMMANDS	8-16

SECTION 8-6: MOVE EXECUTION COMMANDS.....	8-23
SECTION 8-7: MOVE TERMINATION COMMANDS.....	8-25
SECTION 8-8: LOOP CONTROL COMMANDS.....	8-27
SECTION 8-9: HOME AND INITIALIZATION CONTROL COMMANDS	8-31
SECTION 8-10: MOVE SYNCHRONIZATION COMMANDS.....	8-34
SECTION 8-11: SYSTEM STATUS REQUEST COMMANDS.....	8-39
SECTION 8-12: COMMANDS FOR USER UNITS	8-43
SECTION 8-13: POSITION MAINTENANCE COMMANDS	8-44
SECTION 8-14: SLIP AND STALL DETECTION COMMANDS.....	8-48
SECTION 8-15: ENCODER TRACKING COMMANDS.....	8-49
SECTION 8-16: ENCODER STATUS REQUEST COMMANDS	8-50
SECTION 8-17: VELOCITY STAIRCASE COMMANDS	8-52
SECTION 8-18: CONSTANT VELOCITY CONTOURING.....	8-55
SECTION 8-19: COMMAND SUMMARY.....	8-60
SECTION 8-20: APPLICATION PROGRAM EXAMPLES	8-64
8-20-1: RASTER SCAN.....	8-64
8-20-2: REPEATING PROGRAM AND PROGRAMMED OUTPUTS	8-65
8-20-3: LINEAR INTERPOLATION - STAR	8-67
8-20-4: THREE CONCENTRIC CIRCLES.....	8-68

CHAPTER 9: HOST SOFTWARE

SECTION 9-1: DEMONSTRATION SOFTWARE	9-1
SECTION 9-2: README.DOC.....	9-2
SECTION 9-3: BASIC PROGRAM DEMONSTRATION	9-3
SECTION 9-4: PASCAL DEMONSTRATION PROGRAM	9-4
SECTION 9-5: C DEMONSTRATION PROGRAM WITH INTERRUPTS.....	9-16
SECTION 9-6: U14_POLL.C.....	9-27
SECTION 9-7: C DEMONSTRATION PROGRAM WITH DMA.....	9-35
SECTION 9-8: JOG.C	9-41

SERVICE AND REPAIR INFORMATION

INDEX

CHAPTER 1: INTRODUCTION

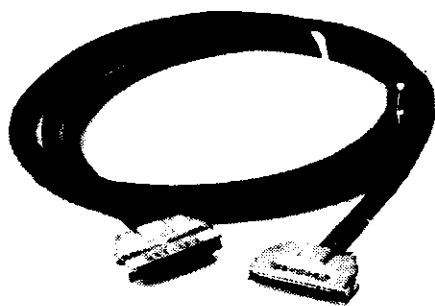
SECTION 1-1: GENERAL DESCRIPTION

The Unidex 14 Motion Control system provides the PC/XT/AT, PS/2-30, and other compatible personal computer users the ability to integrate motion control onto the PC Bus. It consists of a PC-bus Indexer, a Drive chassis, PC to Drive Cabling and software support. It is fully compatible with Aerotech motors and positioning stages.

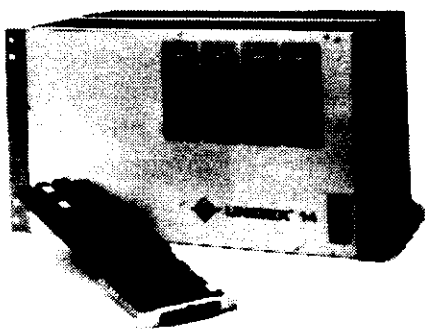
The Unidex 14 Indexer card occupies one full slot on the PC Bus and meets all IBM I/O standards. All programming languages with I/O capability such as Basic, C, and Pascal may be used to program the Unidex 14. Aerotech provides software support on disk with helpful utilities which may be incorporated into the users application programs.

The Unidex 14 provides one to four axis point-to-point and linear interpolation as well as two axis circular interpolation. Motions may be programmed to be performed either absolute or incrementally. Auxiliary outputs can be turned off and on while contouring, making the Unidex 14 suited for many special applications. With sinusoidal acceleration and deceleration the Unidex 14 has virtually eliminated jerk during positioning.

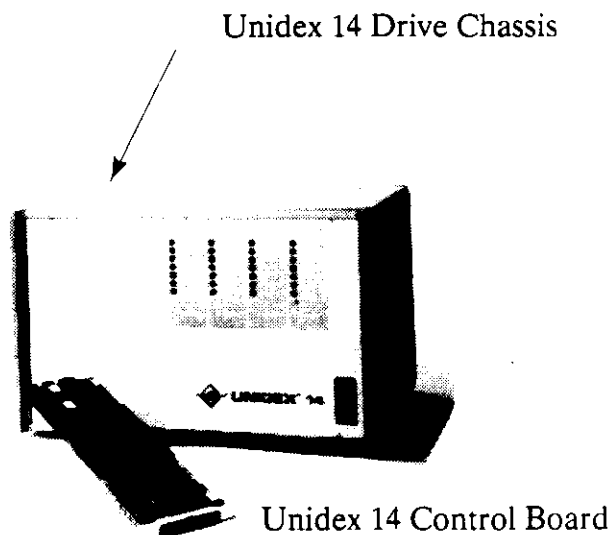
The Unidex 14 Drive Chassis is available in two packages; a 6U - 12" high desktop cabinet - Model U14S and a 6U - 19" rack mount chassis - Model U14R. The PC-Bus card is available independently as Model U14C. These configurations are shown in Figure 1-1. Both configurations are capable of DC, AERODRIVE and Micro-Step drive control. Any combination of DC, AERODRIVE or Micro-Stepping drives may be accommodated.



Unidex 14 Control Board/ Drive Chassis Interconnect Cable



Unidex 14 Model U14R



Unidex 14 Model U14S

Figure 1-1: Unidex 14 Motion Controllers

SECTION 1-2: SPECIFICATIONS

1-2-1: FUNCTIONAL SPECIFICATIONS**MEASURES**

Resolution	
Stepping Models	4,000 steps/rev. (0.09") standard; 400 to 50,000 steps/rev. (0.9" to 0.0072") optional
Servo & AERODRIVE Models	2,000 steps/rev. (0.18") standard; 200 to 4,000 steps/rev. (1.8" to 0.09") optional
Accuracy	
Stepping Models	+/- 5 arc min. typical, unloaded, bidirectional
Servo & AERODRIVE Models	+/- 5 arc min. typical, loaded, bidirectional
Repeatability	+/- 5 arc sec. typical unloaded, unidirectional approach
Hysteresis	3 arc min. or less, unloaded, bidirectional (Stepping models only)

MOTIONS

Axes	1 to 4 (designated X,Y,Z,T)
Type	Linear Interpolation and point-to-point (four of four axes); circular interpolation (any two axes); synchronous or multitasking; velocity profiling; velocity streaming; free-run
Position Range	8 1/2 digit position register
Stepping Models	+/- 16,750 rev. with standard resolution
Servo & AERODRIVE Models	+/- 33,500 rev. with standard resolution
Velocity Range	
Stepping & AERODRIVE Models	0.03 to 3000 rpm (0.0005 to 50 rev./sec.) with standard resolution
Servo Models	0.03 to 5000 rpm (0.0005 to 83 rev./sec. with standard resolution
Acceleration Ramp	0 to 8,000,000 pulses/sec./sec.
Acceleration Profiles	Linear, parabolic, or sinusoidal (programmable)
Positioning Modes	Absolute, incremental (programmable)

PROGRAMMING

Memory(user)	200 character command buffer for each axis; 124 character input buffer
Loop/Wait commands	Loop specified number of times; loop until input LOW; loop until input commanded from host PC; wait until input HI; wait specified time
Scaling	Per axis multiplier permits scaling in metric, english or rotary units
Other Features	Output on-the-fly while contouring; encoder verification (stepping)

INTERFACES

Command	PC/XT/AT interface, meets IBM I/O channel signal specifications and definitions; indexer card occupies one full I/O slot;
---------	---

	I/O address block is user selectable; Interrupt vector is user selectable for levels 2 through 7
Inputs/Outputs	
Dedicated	Buffered inputs per axis for cw, ccw, home limits, marker, amplified sine or line driver encoder
Programmable	5 inputs, optoisolated; 4 outputs, optoisolated; 4 auxiliary outputs, optoisolated (servo and AERODRIVE models only)
Serial Output	Optional serial output (SEO) makes available optoisolated clock, direction, and marker commands (stepping) or cw, ccw clock, and marker (servo and AERODRIVE) for external use.
Brake Control	Optional Output (Opto 22* OD5, rated for 60 VDC @2A) to control electromagnetic brake
CHASSIS WEIGHTS	
U14C	Bus card only: 1 lbs. (0,5kg)
U14S	Stepping & AERODRIVE: 36 lbs. (16,5 kg); D-C Servo: 50 lbs. (23 kg)
U14R	Stepping & AERODRIVE: 30 lbs. (13,5 kg); D-C Servo: 42 lbs. (19 kg)
U14H	Stepping & AERODRIVE: 45 lbs. (20,4kg); D-C Servo: 57 lbs. (25,9kg)
ENVIRONMENTAL	
Ambient Temperature	
Operating	0° to 50° C (32° to 132°F)
Storage	-20° to 70° C (-4° to 158°F)
Humidity	0 to 95%, noncondensing
POWER INPUT	
Driver Chassis	115 VAC, 50/60 Hz. (nom.), 1,000 VA (max.), single-phase
	Optional 230 VAC (nom.) or 100 VAC (NOM.), 50 Hz.
PC-Bus Indexer	+ 5 VDC @ 1.75 amps (nom.) from PC/XT/AT bus
Optoisolated I/O	+ 5 VDC @ 0.15 amps

To: U14s, U14R and U14H chassis (J39)

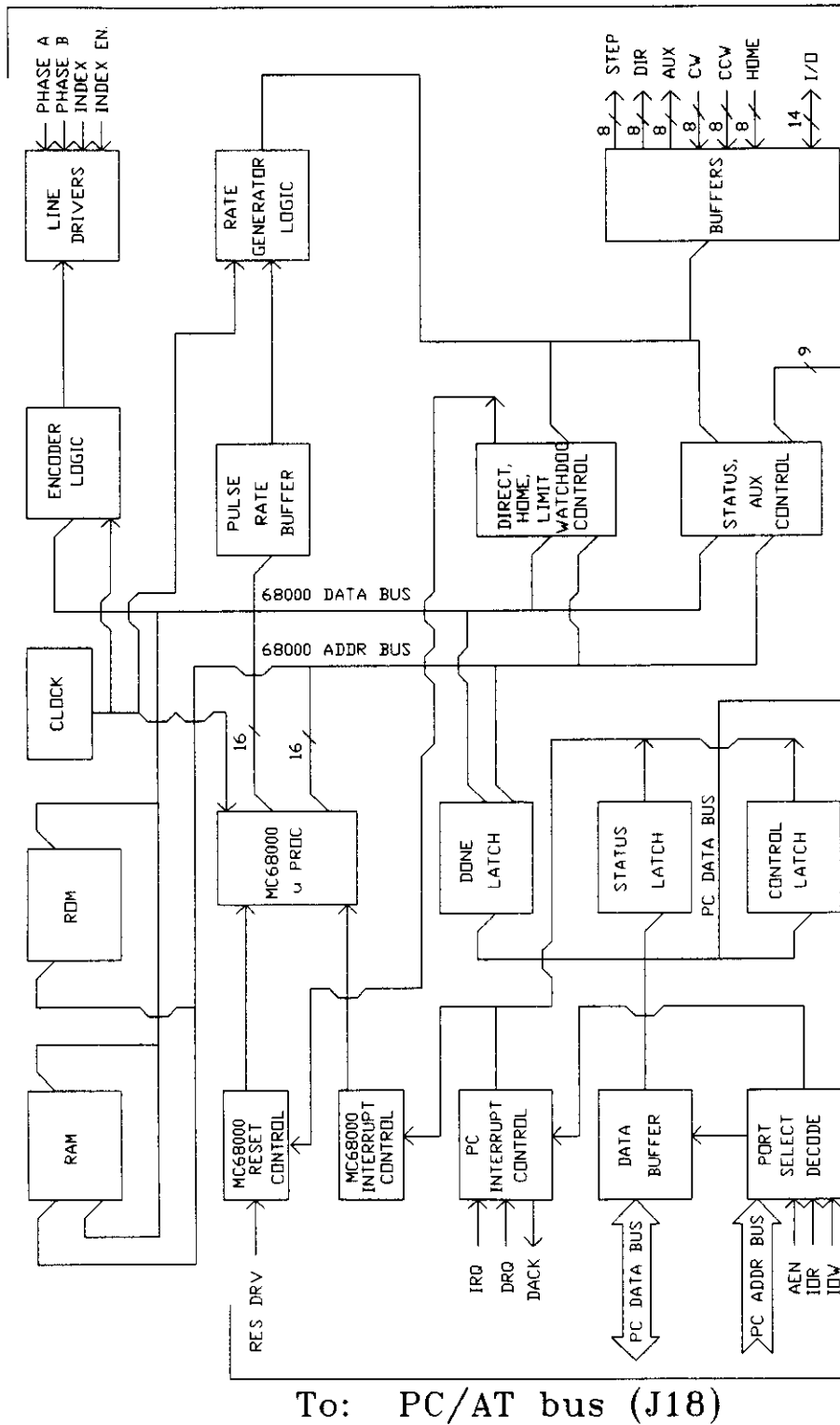


Figure 1-2: Unidex 14 Control Board Block Diagram

CHAPTER 2: GETTING STARTED

SECTION 2-1: INTRODUCTION

The Unidex 14 Motion Controller can control as many as 4 axes of motion from one I/O slot of a PC/XT/AT or compatible computer. It meets the IBM I/O channel specifications and can be interfaced directly into the backplane of these machines. The Unidex 14 can for example simultaneously control four axes of open loop stepping motors while monitoring their actual position with the built in incremental encoder interface. The Unidex 14 can be set to correct the stepping motors to the desired position at the end of the indexers move. In addition "closed loop" DC Servo Motors and Aerodrive Servo Modules can be installed in the U14 driver chassis to provide closed loop brush or brushless motor control. A Unidex 14 can manage coordinated or independent motion on each of the four axes simultaneously.

The Unidex 14 functions as a motion coprocessor within the PC/XT/AT or compatible computer. It utilizes a 68000 microprocessor and patented proprietary technology to control direction of motion, acceleration, deceleration and velocity of an associated motor. In response to commands from the host computer, the Unidex 14 controller will calculate the optimum velocity profile to reach the desired destination in the minimum amount of time while conforming to the programmed velocity and acceleration parameters. A block diagram of the Unidex 14 Control Board is shown in Figure 1-2.

Commands may be sent to the Motion Controller by simple I/O commands using virtually any language on the PC which has the ability to do I/O. These controllers are easily programmed using ASCII character strings. For a typical motion requirement of 1,000,000 pulses at 400,000 pulses/sec and an acceleration of 500,000 pulses/sec/sec the following string would be sent from the host computer to the Unidex 14:

VL400000 AC500000 MR1000000 GO

For additional programming examples see Section 8.

SECTION 2-2: THEORY OF OPERATION

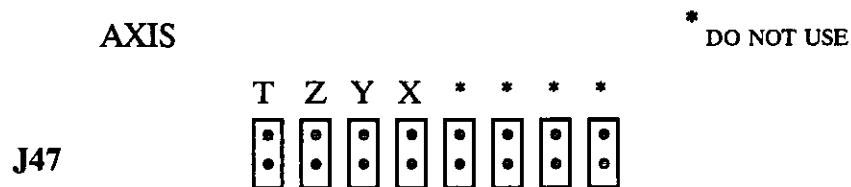
The 68000 microprocessor of the Unidex 14 controller maintains four concurrent processes. The highest priority process calculates the desired pulse frequency 1024 times each second with a proprietary algorithm (patent number 4,734,847). This number is fed to U62, U63, U64 and U65 which synchronizes the motor pulses to the microprocessor and actually generate the pulse train. The velocity profile and synchronization of each axis is also handled by the 68000. (See Section 2-5 for more information).

The commands from the PC/XT/AT or compatible host computer are temporarily stored in a 124 character buffer until the 68000 microprocessor can parse them. The command is then executed immediately or routed to separate command queues for each axis. The command queue contains a list of addresses to execute followed by an optional parameter. A command from the host may be expanded into several commands to the appropriate axis. The GO command for example will expand into start, ramp up, constant velocity and ramp down commands. The LS command will save its parameter, i.e. the loop count, on a loop stack along with the address of the LS command to be used by the next LE command as a target for a jump command. The LE command will decrement the loop count and jump to the most recent LS command providing the loop count has not reached zero. If the loop count has reached zero and it is not nested inside another loop, the queue space will be flagged as available and the next instruction in the queue will be executed.

Interrupts to the PC/XT/AT host are latched by U33. (Refer to Figures 5-2 through 5-21 Circuit Schematics of the U14C Control Board.) The enable status of each interrupt is also stored by U33. Status of the interrupts and error flags may be read by the host via U33. U12 latches the interrupt request enable bits and allows them to be read back by the microprocessor. U16 compares the Unidex 14 address to the I/O address selected by the host and enables the board decode logic when a match is detected.

The U14C board requires one full length slot in the PC/XT/AT. **In most cases the jumpers on the U14C board will not have to be changed if there are no conflicts with interrupt and I/O address lines.** The factory default settings for the board provide for I/O addresses 300 to 303 (hex) and the IRQ5 interrupt line. If these do not conflict with any previously installed hardware in your computer then you will not need to change any jumpers on the U14C board. There are seven blocks of square pin jumpers on the U14C board J47, J15, J17, J67, J57, J19 and JP79. See Figure 2-1 for the location of the jumpers.

J47 determines whether the limit inputs to an individual axis are active low or active high. (Note: If limits polarities are to be changed, see also Chapter 6 Drive Modules.) With the jumper in place, the associated axis will stop moving if the limit line, for the direction the axis is moving, is grounded. With the jumper removed, the axis will stop if the limit line is at +5 volts. These lines are internally pulled-up with a 2.2K ohm resistor to +5 volts so that only a switch is required to control the limit lines.



Limit Sense Jumpers for Four Axes Board

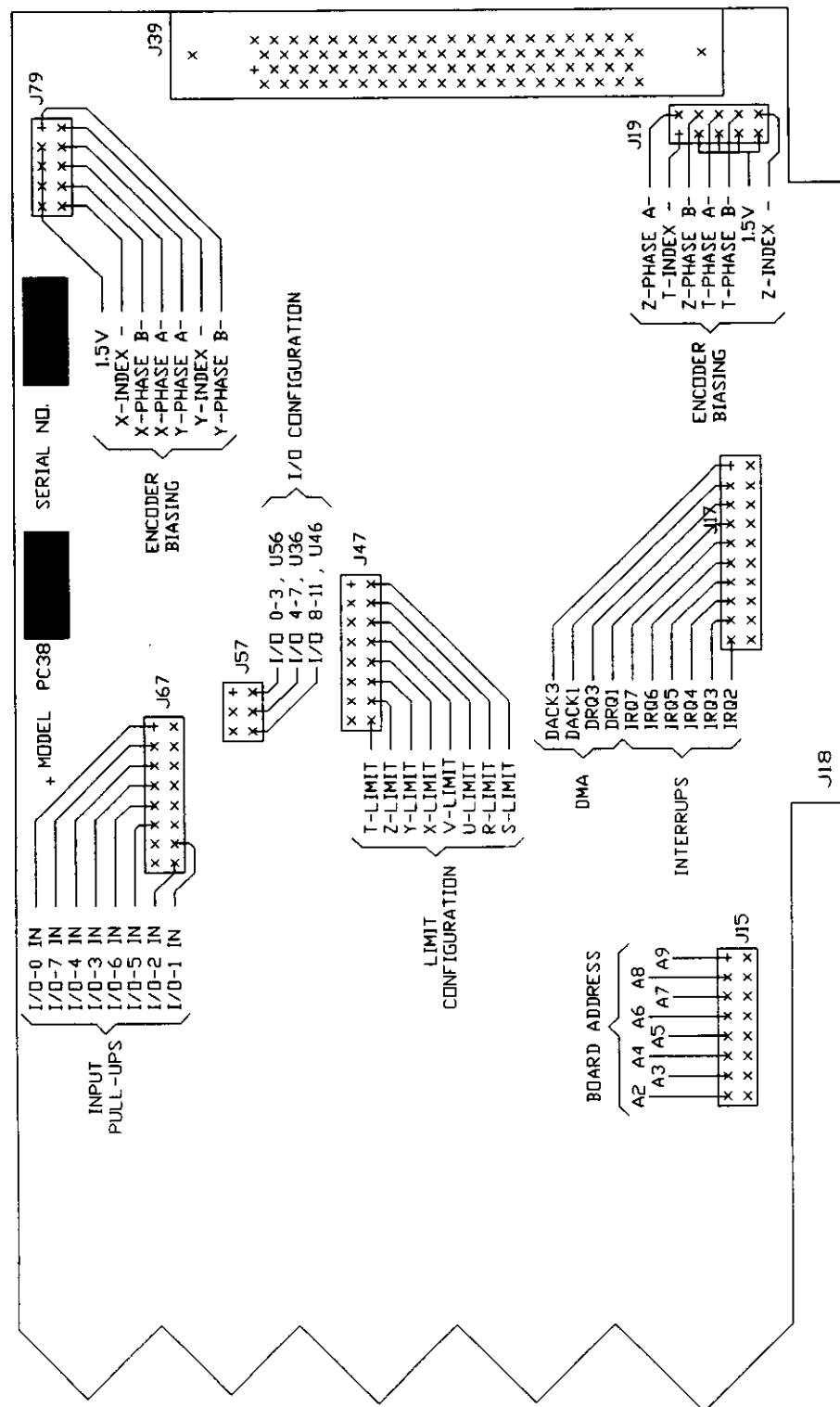
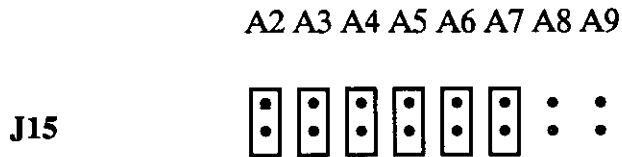


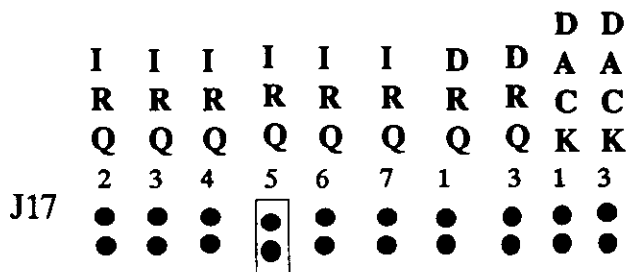
Figure 2-1: Location of Option Jumpers 6U U14 Control Board

J15 determines the I/O address range of the board. A jumper across a pair of pins sets that bit in the address low. With the jumper removed the bit is high. The computer's I/O address ranges from 200 to 3FF (hex). The A0 and A1 address lines of the PC are decoded internally by the Unidex 14 board and are assumed to be 0 for base address calculations. The jumpers set the base address lines A2 through A9. The factory default address is 300 hex. This requires jumpers across A2 and A3, making all four of the lowest bits a 0 and the least significant hex digit of the address a 0. Jumpers across A4, A5, A6 and A7 make the middle four bits 0, making the middle hex digit of the address a 0. Removing the jumpers from A8 and A9 makes each of these lines a 1, making the most significant hexadecimal digit a 3.



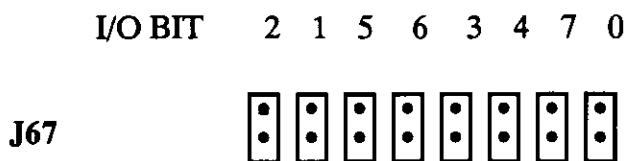
Address Select Jumpers (Default Setting)

J17 determines which interrupt level the U14C board will use when it communicates. IRQ5 is the factory default setting. This jumper block also controls which DMA control lines are used by the board, if DMA is used by the controlling program. The DMA lines are unused in most applications. The program U14_CDMA.C on the DEMO disk and listed in the software section uses DRQ1 and DACK1 lines. If that program is used, jumpers need to be placed across those sets of pins.



Interrupt and DMA Jumpers

J67 connects I/O bits 0-7 to pull-up resistors when they are configured as inputs. J57 is directly below J67 and selects the configuration of I/O bits 0-11 as inputs or outputs. A jumper on pins 3 and 4 of J57 selects I/O bits 0-3 as inputs. Note that U56 must be a 7402 when these pins are configured as inputs. No jumper on pins 2 and 5 of J57 selects I/O bits 8-11 as outputs and U36 must be a 7400 or 7438 open collector gate. Figure 2-2 shows the required configuration for an input bit while Figure 2-3 shows the configuration for an output bit. (See also Figure 5-21)



User I/O Pull-Up Jumpers (Default Setting)

J79 and J19 are for biasing unused encoder inputs. When a single end encoder is used, the unused inputs can be biased using wire-wrap wire or jumpers. J79 is for the X and Y axes and J19 is for the Z and T axes. See Figure 2-1 for jumper locations.

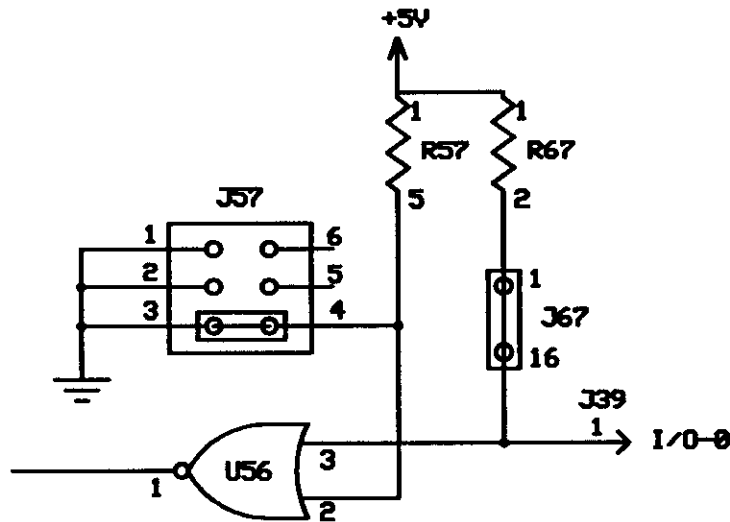


Figure 2-2: User I/O Input Configuration

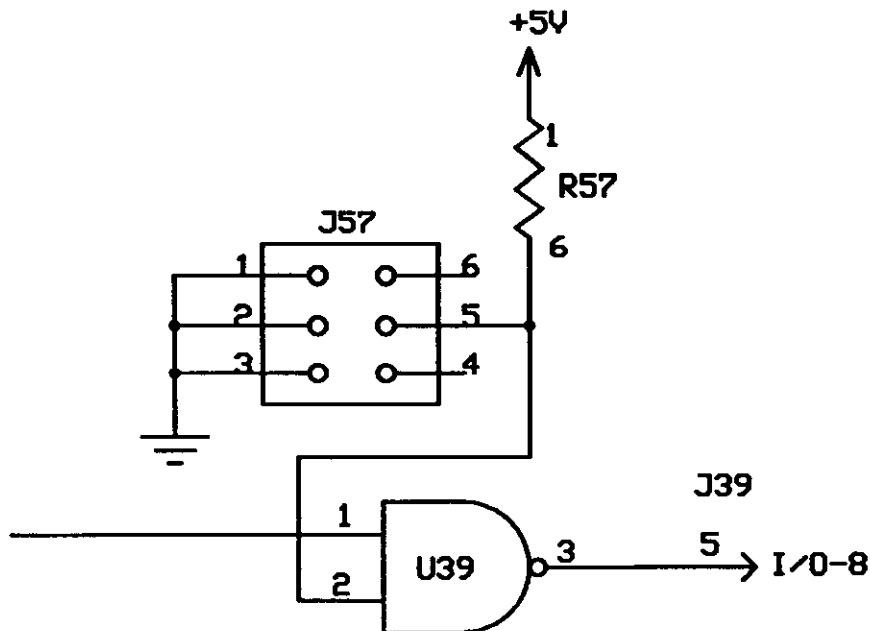


Figure 2-3: User I/O Output Configuration

2-3-2: HARDWARE INSTALLATION FOR U14 CONTROL BOARD

1. Turn the computer power off and disconnect its power cord from power source.
DANGER: PRIOR TO INSTALLING THE U14C. MAKE CERTAIN ALL POWER CONNECTIONS ARE DISCONNECTED.
2. Remove the computer's cover.
3. Choose an empty expansion slot in the mother board and remove its associated metal cover from the back of the computer. Be sure to save the screw.
4. Check the U14C board's jumpers for proper configuration.
5. Slide the U14C board down into the edge connector, insuring the board is lined up correctly in the card guides and firmly in the connector.
6. Use the screw removed from the cover to attach the metal bracket of the U14C board to the chassis of the computer.
7. Make certain that the board it is properly seated in the connector.
8. Replace the cover of the computer.
9. Replace the power cord and turn the computer on. (For ease of trouble-shooting do not connect U14C to other parts of system until communication is established with the host.)
10. Allow the computer to boot up.
11. When boot-up is complete, install the Support Disk into the floppy disk drive. If the factory default jumper settings are being used, type: A:U14_BAS. This loads the program, and allows the User to enter Unidex 14 control commands at the keyboard or to download a file of commands. The responses from the U14C board will be displayed on the computer's screen.
Display the "README" file for information pertaining to the most current Unidex 14 software.

NOTE: The README file may contain software support information, not contained in this Manual, that is pertinent to Unidex 14 operation.

12. To enter the terminal emulation mode without downloading a command file, press a Carriage Return when the program asks for a file name.
13. The program will state that you are in the interactive mode. Enter the Unidex 14 commands EN WY on the keyboard. EN turns echoing on which causes the typed commands to be echoed from the controller to the display. WY asks the U14C board "who are you". The board responds with its model type and firmware version number. (i.e. PC38 ver 1.39-4E)
14. If a similar message is displayed, the board was correctly installed and communication is established with the Unidex 14.
15. If no similar message is displayed, verify the installation procedure, if the problem cannot be determined, contact the Customer Service Department at AEROTECH, Inc. for assistance.

2-3-3: SOFTWARE INSTALLATION

The Support Disk contains several versions of a program which allow the User to interact with an AEROTECH U14C control board. The User may type Unidex 14 commands on the computer's keyboard and they are passed to the U14C board. The U14C board's responses are displayed on the computer's screen. The User may, when prompted, optionally give the program the name of an ASCII text file that contains Unidex 14 commands. The program will send the contents of the file to the U14C board.

All C language programs were compiled using either Microsoft Quick C or Microsoft C Version 5.0 Optimizing Compiler. Older versions of this compiler and C compilers from other companies may not compile this program properly because of it's use of hardware interrupts. The program U14_BAS.BAS was compiled using the Microsoft QuickBASIC Version 4.5 compiler.

Most of the programs expect the U14C board to be jumpered to the factory default I/O address of 300 HEX. The "C" versions use IRQ5 to communicate with the U14C board. This is the factory default setting. The BASIC and Pascal versions do not use interrupts, instead they use polling.

Also on this disk are the following: a program U14_CDMA.C that uses the PC/XT/AT DMA controller to send commands to the U14C board. If this program is executed, the appropriate jumpers on the U14C board will have to be configured.

The source code for these programs is included on the disk allowing Unidex 14 Users to use the routines in application programs. No license is required. The programs are also listed in the software section of this manual. Refer to the software Support Disk provided for the most current example programs (this manual may not be revised as quickly as the software Support Disk).

2-3-4: MOTOR CONTROL CONNECTOR

The motor control connector (J39) on the U14C board consists of two rows of square pins, each row has 40 pins for a total of 80 pins. Fourteen of the pins are User definable I/O which can be used to synchronize motor movement to external events. (See Figure 2-3A for "Pin-Out" specifications.)

The other 64 pins can be considered as 4 logical sets of 16 pins. Each set is used for an individual axis.

The 16 pins of an axis set are as follows: Step Output, +5 Volts Output, Auxiliary Output, Direction Output, Clockwise Limit Switch Input, Ground, Home Switch Input, Counter-Clockwise Limit Switch Input, and, Index +, +5 Volts, Phase A -, Phase A +, Index -, Ground, Phase B -, Phase B +.

NOTE: The User typically need not concern himself with these Pin Out specifications since the "Interconnect Cable" shown in Figure 1-1 is provided for interconnection between the J14 Control Board and Drive Chassis.

DESCRIPTION	PIN	PIN	DESCRIPTION
User I/O 0	1	41	+5 volts
User I/O 2	2	42	User I/O 1
User I/O 4	3	43	User I/O 3
User I/O 6	4	44	User I/O 5
User I/O 8	5	45	User I/O 7
User I/O 10	6	46	User I/O 9
User I/O 12	7	47	User I/O 11
User I/O 13	8	48	Ground
X phase A+	9	49	+5 volts
X phase A-	10	50	X index +
X phase B+	11	51	Ground
X phase B-	12	52	X index -
Y phase A+	13	53	+5 volts
Y phase A-	14	54	Y index +
Y phase B+	15	55	Ground
Y phase B-	16	56	Y index -
X direction output	17	57	+5 volts
X auxiliary output	18	58	X step output
X CCW limit switch	19	59	Ground
X home switch	20	60	X CW limit switch
Y direction output	21	61	+5 volts
Y auxiliary output	22	62	Y step output
Y CCW limit switch	23	63	Ground
Y home switch	24	64	Y CW limit switch
Z direction output	25	65	+5 volts
Z auxiliary output	26	66	Z step output
Z CCW limit switch	27	67	Ground
Z home switch	28	68	Z CW limit switch
T direction output	29	69	+5 volts
T auxiliary output	30	70	T step output
T CCW limit switch	31	71	Ground
T home switch	32	72	T CW limit switch
Z phase A+	33	73	+5 volts
Z phase A-	34	74	Z index +
Z phase B+	35	75	Ground
Z phase B-	36	76	Z index -
T phase A+	37	77	+5 volts
T phase A-	38	78	T index +
T phase B+	39	79	Ground
T phase B-	40	80	T index -

Figure 2-3A: J39 Encoder Input and I/O Pin Assignment

SECTION 2-4: UNIDEX 14 INTERFACE BOARD

2-4-1: JUMPER CONFIGURATIONS

NOTE: Aerotech Inc. has configured the Unidex 14 at the time of assembly, per Customer specifications. The following sections provide jumper configuration information to be used only if the system specifications have changed since the time of manufacture.

Refer to Figure 2-4 for Unidex 14 Interface Board Jumper information.

2-4-1-1: INPUT VOLTAGE

The desired AC input voltage may be configured by the following jumper configurations:

For 115 VAC (standard) operation -

Insert JP1,JP3

Remove JP2,JP4

For 230 VAC (optional) operation -

Insert JP2,JP4

Remove JP1,JP3

2-4-1-2: OPTIONAL ENCODER FEEDBACK

If encoder feedback to the Unidex 14 is desired it must be jumpered for either an analog sine wave or a TTL logic encoder.

All axes with an analog sine wave encoder must have a sine wave encoder terminal board (Aerotech dwg. no. 690D1333) installed. Axes jumpers must be configured as follows for an analog sine wave encoder or TTL logic encoders as follows:

X Axis	Sine Wave	Logic
JP20	connect 1-2, 2-3 removed	connect 2-3, 1-2 removed
JP21	"	"
JP22	"	"
JP23	"	"

Y Axis	Sine Wave	Logic
JP30	connect 1-2, 2-3 removed	connect 2-3, 1-2 removed
JP31	"	"
JP32	"	"
JP33	"	"

Z Axis	Sine Wave	Logic
JP40	connect 1-2, 2-3 removed	connect 2-3, 1-2 removed
JP41	"	"
JP42	"	"
JP43	"	"

T Axis	Sine Wave	Logic
JP50	connect 1-2, 2-3 removed	connect 2-3, 1-2 removed
JP51	"	"
JP52	"	"
JP53	"	"

2-4-1-3: INPUT/OUTPUT BITS OR SYSTEM CONTROL

2-4-1-3-1: I/O BIT 5 OR AT HOME

I/O Bit 5 may be used as an Opto-Isolated INPUT by configuring JP17 with 2-3 connected, 1-2 removed. I/O Bit 5 may also be used as an indicator as to when all axes have completed their Home cycle (see I/O Bit 13) by configuring JP17 with 1-2 connected, 2-3 removed.

2-4-1-3-2: I/O BIT 6 OR FAULT

I/O Bit 6 may be used as an Opto-Isolated INPUT by configuring JP6 with 2-3 connected, 1-2 removed. I/O Bit 6 may also be used to sense Drive fault conditions by configuring JP6 with 1-2 connected, 2-3 removed. Once configured in this manner, if any of the four axis faults, this bit becomes true. See Chapter 6 for a description of the Drives having a Fault Output.

2-4-1-3-3: I/O BIT 7 OR CZ INPUT (IN POSITION)

I/O Bit 7 may be used as an Opto-Isolated INPUT by configuring JP16 with 2-3 connected, 1-2 removed. I/O Bit 7 may also be used as an indicator as to when all of the Servo Drives have completed their move by configuring JP16 with 1-2 connected, 2-3 removed. Once configured in this manner, when all Servo Drives have completed their move, this bit becomes true.

LO/HI or the CZ Select Jumpers 12-15 must be configured appropriately in accordance with Item 2-4-1-5 when I/O Bit 7 is configured in this manner.

2-4-1-3-4: I/O BIT 12 OR LOCAL/REMOTE

I/O Bit 12 may be used as an Opto-Isolated OUTPUT by configuring JP7 with 2-3 connected, 1-2 removed. Also JP18 with 1-2 connected, 2-3 removed, JP19 with 2-3 connected, 1-2 removed sets the standard active state which may be changed by setting JP19 with 1-2 connected, 2-3 removed. I/O Bit 12 may also be configured to select operation of the drives from a Remote Clock and Direction source through the Auxiliary connectors (per axis) on the rear of the Unidex 14 Interface Board. I/O Bit 12 is configured in this manner by setting JP7 with 1-2 connected, 2-3 removed, JP18 with 2-3 connected, 1-2 removed and JP19 with 1-2 connected, 2-3 removed.

The Local mode must be active to control the axis from the Unidex 14's Clock and Direction commands (this is the default state upon power up).

2-4-1-3-5: I/O BIT 13 OR GO HOME (HARDWARE HOME)

I/O Bit 13 may be used as an Opto-Isolated OUTPUT by configuring JP5 with 2-3 connected, 1-2 removed. I/O Bit 13 may also be configured to trigger the Driver's hardware home circuit of all axes simultaneously by setting JP5 with 1-2 connected, 2-3 removed.

2-4-1-4: AXIS AUXILIARY OUTPUTS (LO/HI CURRENT CONTROL)

Each axis (X,Y,Z,T) has an Auxiliary Output (JP11, JP10, JP9 and JP8 respectively) which may be used as an Opto-Isolated OUTPUT. The Auxiliary Outputs are synchronized to each axes motion by configuring the appropriate jumper with 1-2 connected, 2-3 removed.

LO/HI or the CZ Select Jumpers 12-15 must be configured appropriately in accordance with Item 2-4-1-5 when the I/O Bit 7 is configured in this manner. Axis Output may also be used to toggle Stepping Motors to the "Stand-By" (Lo Current) mode when at rest by setting the axes jumper with 2-3 connected, 1-2 removed.

2-4-1-5: LO/HI CURRENT CONTROL OR CZ SELECT (IN POSITION)

Each axis (X,Y,Z,T) has a jumper (JP15, JP14, JP13,JP12 respectively) for a multi-purpose signal on the Drive module. If an axis Drive module is a Stepper, the respective jumper should be configured with 1-2 connected, 2-3 removed. (See also Item 2-4-1-4.) If an axis Drive module is a DC Servo then the axes respective jumper should be configured with 2-3 connected, 1- 2 removed. (See Item 2-4-1-3-3.)

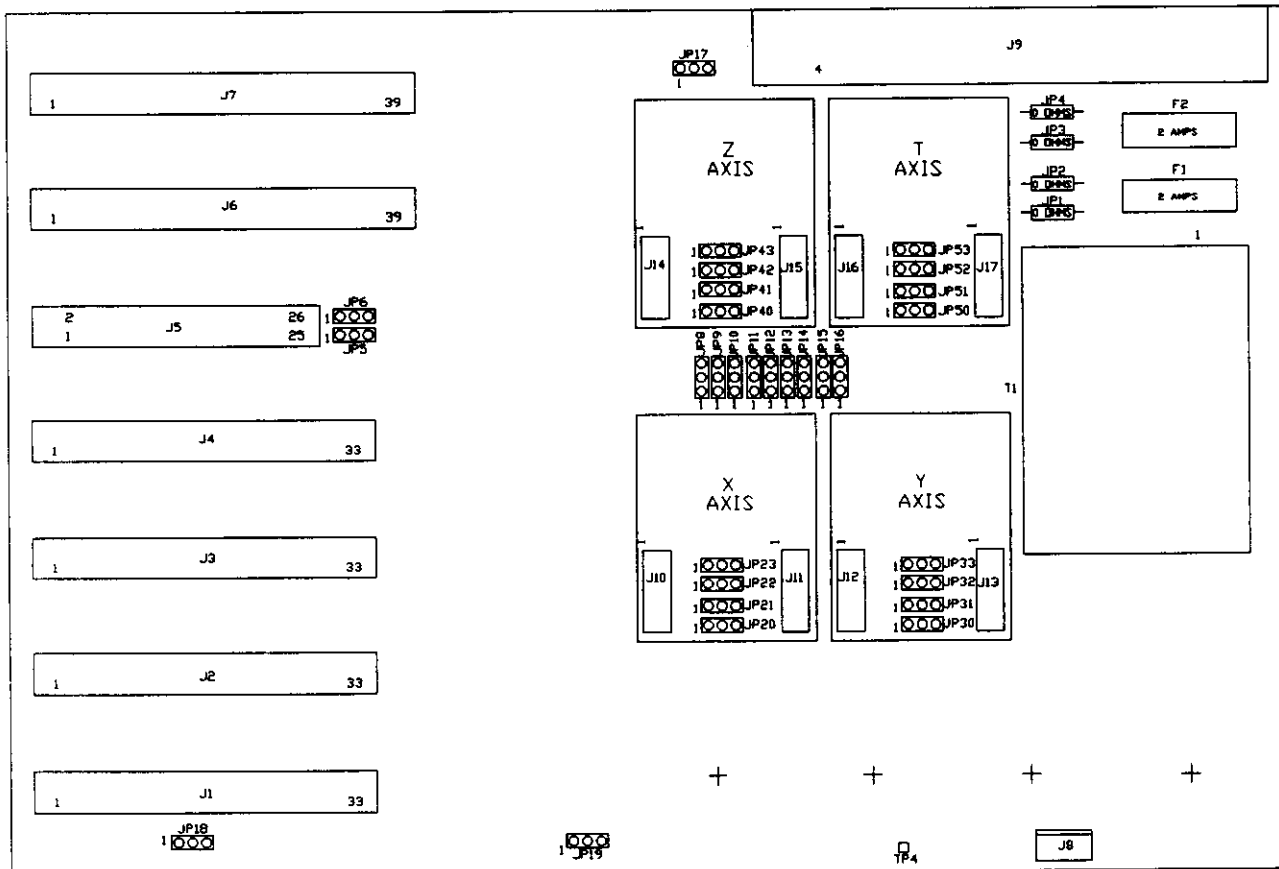


Figure 2-4: Unidex 14 Interface Board

SECTION 2-5: VELOCITY PROFILES

2-5-1: FUNCTIONAL DESCRIPTION

The Unidex 14, in response to commands from the host computer, provides controlled acceleration to a predefined peak speed followed by calculating the optimum velocity 1024 times each second providing a very smooth acceleration curve. This calculation is used to control a variable frequency pulse train which is derived from a crystal oscillator providing very accurate pulse rates.

The 68000 microprocessor calculates this velocity profile for each of the axes providing independent but synchronized (if desired) profiles for each axis. The Unidex 14 can perform a smooth coordinated move on all four axes using linear, parabolic or cosine velocity profiles. It can manage four independent or coordinated processes.

The Unidex 14 will calculate the optimum velocity profile to generate the desired move while conforming to the acceleration and velocity data input by the host computer. This move will consist of a smooth acceleration followed by a constant velocity section and a smooth deceleration to the desired distance. A graph of a typical linear velocity is shown in Figure 2-5.

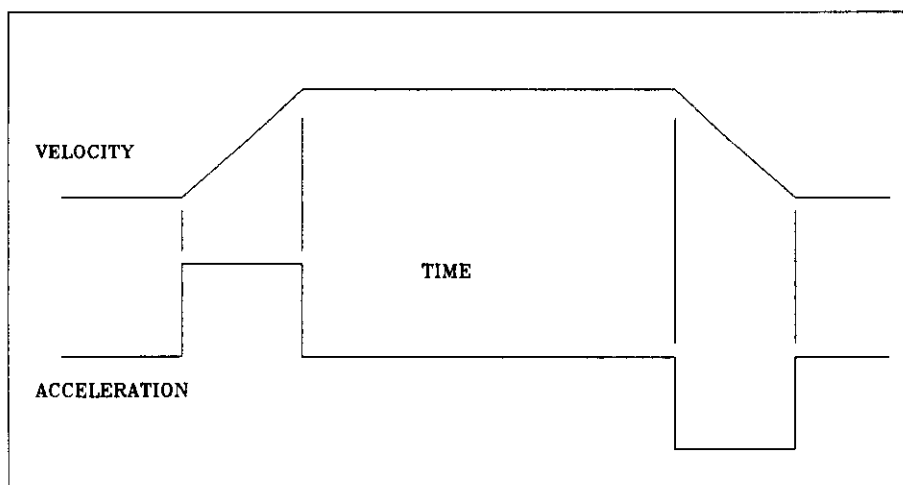


Figure 2-5: Typical Velocity Profile

If the move parameters do not allow the motor to accelerate to the desired velocity in the desired distance, the Unidex 14 will automatically generate an optimum triangular velocity profile. It can also be commanded to accelerate to a velocity and hold that velocity until told to stop or change to a new velocity. It will then smoothly decelerate to a stop or accelerate/decelerate to the new velocity.

Several moves of this type may be chained together to provide a more complex pattern. The Unidex 14 is able to store up to 124 characters in an input character buffer and 200 commands and parameters in separate command queues for each axis allowing several moves to be made without host intervention. A loop counter is provided to repeat desired sections of a complex move pattern. Loops may be nested up to four levels deep on all axes.

2-5-2: VELOCITY PROFILE OPTIONS

The Unidex 14 Motion Controller offers three options for ramping the device to speed. The traditional constant acceleration or linear velocity ramp (Figure 2-5) is the default on reset or power up. The half sinusoid acceleration or half cosine velocity ramp (Figure 2-7) is selected by the CN command. Since the acceleration is zero at the velocity inflection points, this provides very smooth operation. It is used in sensitive applications such as wafer handling on a vacuum chuck. The third option is a reverse ramp of acceleration or Parabolic Velocity curve (Figure 2-6) which can be selected by the PN command. This ramp is commonly used to compensate for loss of motor torque at high speeds, i.e. since the acceleration is reduced at higher speeds the required forces are reduced proportionally. This type of ramping is suitable for open-loop stepping motor control where as linear and half sinusoid's moves are suitable for DC Brush or Aerodrive control. The parabola may be truncated to allow the user to select, under program control, the reduction in acceleration (force) appropriate for the application.

2-5-2-1: LINEAR RAMPS

The Unidex 14 Motion Controller generates a linear velocity ramp in real time, i.e. while the stage is in motion. There is no table building prior to the move and thus minimal latency. The controls will accelerate to the specified velocity and hold that speed until just enough move distance is left, then decelerate to a stop. If the move distance is too short to reach speed, a triangular velocity ramp will automatically be generated. The acceleration is a constant A_m and the velocity is then:

$$v = A_m t$$

A useful relationship in the distance required to accelerate at acceleration A_m to peak velocity V_p is as follows:

$$s = \frac{\pi V_p^2}{2A_m}$$

or the acceleration A_m required to accelerate to peak velocity V_p in distances s is as follows:

$$A_m = \frac{V_p^2}{2s}$$

2-5-2-2: PARABOLIC RAMPS

The parabolic ramp is generated in a similar fashion except the acceleration is reduced as the stage accelerates to speed thus reducing the velocity slope as shown in Figure 2-6.

The acceleration follows the equation:

$$a = A_0 - A_0 \frac{t}{T}$$

and the velocity is then:

$$v = A_0 t - \frac{A_0 t^2}{2T}$$

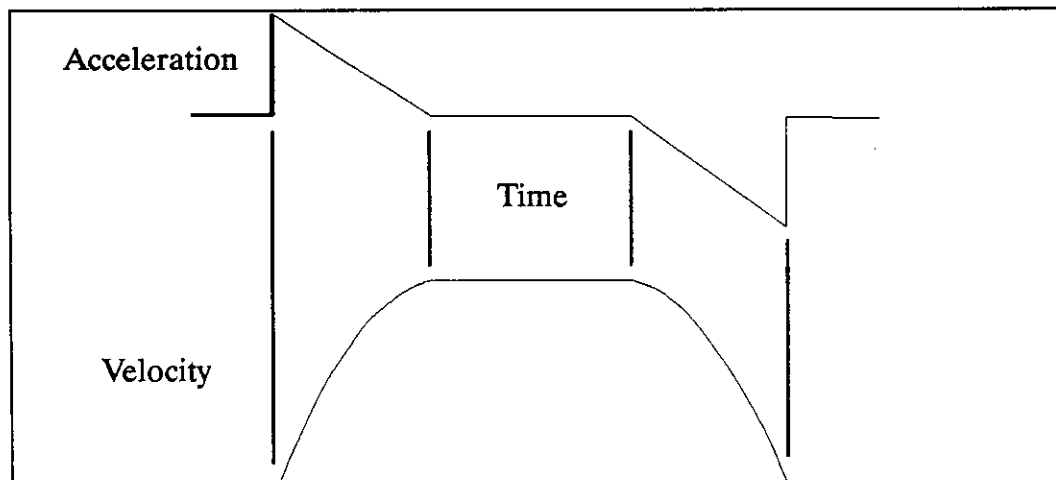


Figure 2-6: Parabolic Velocity Profile

and the distance traveled in the ramp is:

$$s = \frac{A_0 t^2}{2} - \frac{A_0 t^3}{6T_2}$$

where A_0 is the initial acceleration, t is time during the ramp and T_2 is total ramp time if the acceleration $\frac{t}{T_2}$ had reached zero. The parameter supplied with the PN command is 10 times the ratio which can take on values from 3 to 10, allowing the final acceleration to range from 70% to 10% respectively of the programmed or initial value. When a move is specified, the controls will fit the resulting velocity curve to the desired acceleration profile. This insures that the desired acceleration is always reached at the programmed velocity as long as the move is long enough for the motor to reach the programmed speed. If the move is too short to reach the programmed speed the curve is truncated causing the shape of the velocity curve to remain the same up to the velocity reached by the specific move. This is consistent with the desired result of compensating for loss of motor torque since the motor has not reached the programmed speed, less compensation is needed. The parabolic ramp mode may result in reduced move time at high speeds, since a larger acceleration may be used.

2-5-2-3: COSINE RAMPS

The cosine ramps are generated in a similar fashion to the parabolic ramps, except the acceleration is:

$$a = A_m \sin \frac{2A_m}{V_p} t$$

and the velocity is then:

$$v = \frac{V_p}{2} \left(1 - \cos \frac{2A_m}{V_p} t \right)$$

and the distance traveled in the ramp is:

$$s = \frac{V_p}{2} t - \frac{V_p^2}{4A_m} \sin \frac{2A_m}{V_p} t$$

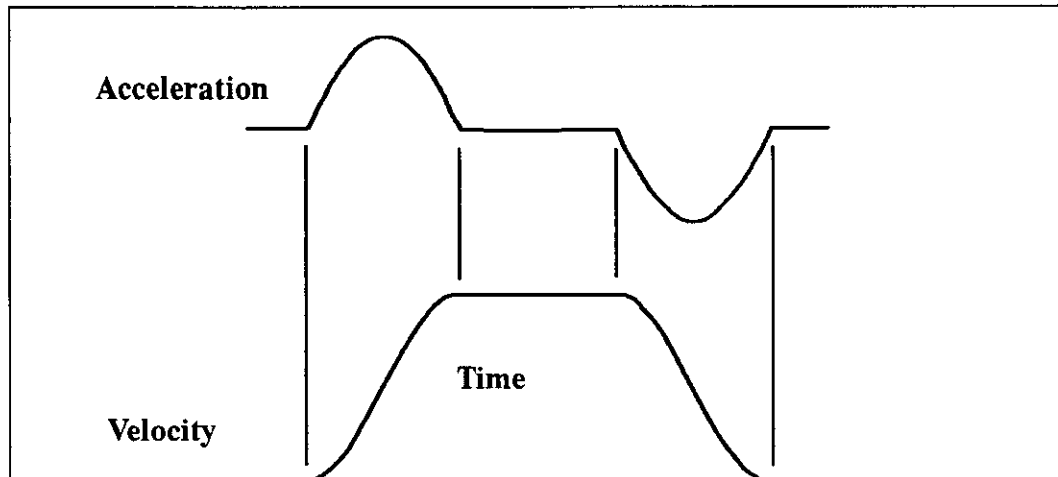


Figure 2-7: Cosine Velocity Profile

where V_p is the peak velocity, A_m is the peak acceleration. The distance needed to ramp up is then:

$$S_1 = \frac{\pi V_p^2}{4A_m}$$

and the time required to ramp up is:

$$T = \frac{\pi V_p}{2A_m} = \sqrt{\frac{\pi S_1}{A_m}}$$

and the peak velocity is:

$$V_p = \sqrt{\frac{4A_m S_1}{\pi}}$$

The cosine ramp requires $\frac{\pi}{2}$ times longer than a linear ramp to reach the same velocity when using the same peak acceleration.

Since the purpose of the cosine ramp is smooth operation, it is desirable to adjust the velocity parameters such that the desired profile is achieved even when the stage does not reach the programmed speed as opposed to truncating the curve as is done with the Parabolic curves.

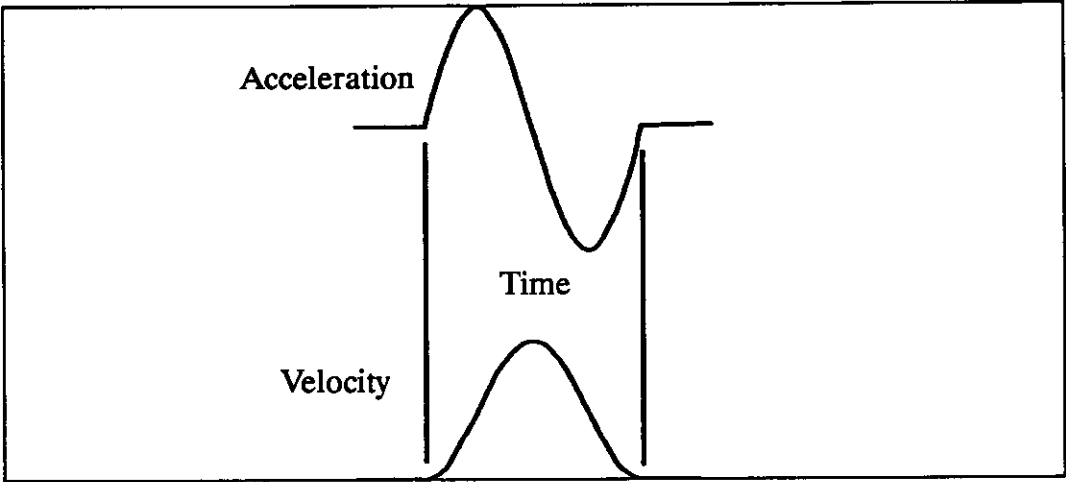


Figure 2-8: Short move Cosine Velocity Profile

CHAPTER 3: I/O CHANNEL INTERFACE

SECTION 3-1: I/O CHANNEL

The PC and XT system bus has available several slots for interfacing I/O cards such as the U14C. The AT has an additional connector to make available the 16 bit data bus and additional address lines. Since the U14C is an I/O device, it uses only the low order 8 bits of the data bus and the first 10 address lines in this family of computers. It can thus be used in any computer which meets the IBM I/O channel specification. The following table shows the I/O channel signals which are also the descriptions of the pins on the edge connector of the U14C board. The front or component side of the board is the A side of the connector of the U14C while the back or solder side is the B side.

3-1-1: PC/XT/AT DATA BUS.

The data bus is an 8 bit, bidirectional, 3-state bus. Direction of data is controlled by the PC/XT/AT host computer. The data bus uses high-level active logic. All data and commands are passed through this data bus. (See Figures 5-3 and 5-4)

3-1-2: PC/XT/AT ADDRESS BUS.

The address bus is a 20-bit high-level active bus. This bus is always driven by the PC/XT/AT host computer. The address bus provides the 20 address lines for decoding by either memory or I/O. (See Figures 5-3 and 5-4)

3-1-3: MEMORY AND I/O CONTROL

These lines provide the signals for fundamental memory and I/O operations. MEMW, MEMR, IOW and IOR control lines distinguish between memory and I/O transfers and determine the direction of transfer. Since the U14C is an I/O device it uses only A0 through A9 and only IOW and IOR to determine the direction of transfer. (See Figures 5-3 and 5-4 and Table 3-1)

3-1-4: INTERRUPT REQUEST

The PC/XT/AT supports interrupts to the system bus at levels 2 (IRQ2) through 7 (IRQ7). These are active high input only signals to the system bus. They go directly to the system interrupt controller which generates the vector during an interrupt acknowledge cycle. The U14C supports all 5 interrupt levels which can be selected by a jumper at J17. Refer to Section 3-3 for details of configuration. (See also Figure 5-4)

Table 3-1 Control Line Description

SIGNAL	DESCRIPTION
MEMW*	This is a low level active signal used to write data from the system bus into memory and is thus not used by the U14C.
MEMR*	This is a low level active signal used to read data from memory to the system bus and is thus not used by the U14C.
IOW*	This is a low level active signal used to write data into I/O. It is driven by the system bus and indicates the address bus contains a valid I/O address. It is used by The U14C as an address qualifier for write operations.
IOR*	This is a low level active signal used to read data from an I/O device onto the system bus. It is driven by the system bus and indicates the address bus contains a valid I/O address. It is used by the U14C as an address qualifier for read operations.
ALE	This output only signal driven by the system bus is decoded by the U14C to indicate a valid address on the system bus.

* Indicates low true signal

3-1-5: DIRECT MEMORY ACCESS

The PC/XT/AT supports DMA levels 1 and 3 on the system bus. DRQ1 through DRQ3 are active high input only request lines while DACK1 through DACK3 are active low level output only signals issued by the system to acknowledge the DMA request. AEN is also decoded by the U14C as required to decode the DMA transfer operation. The U14C supports levels 1 and 3 in either the read or write direction. The level is selected via J17. Refer to Section 3-4, for configuration details. The direction of transfer is under software control, see Chapter 4, for an example program.

3-1-6: CLOCK AND OSC LINES

These are provided on the system bus. The U14C has its own on-board clock and does not use either of these signals.

3-1-7: RESET DRV

This line is a reset driver which is provided on the bus. The U14C has an on board reset timer and uses the RESET DRV signal to initiate this timer on power up or during a system reset. This bus signal is not active during a warm boot such as the CTL-ALT-DEL or CTL-BREAK and thus the U14C is not reset. It can, however be reset on computers equipped with a reset push button.

3-1-8: I/O CH RDY

This line is used in conjunction with memory wait state generation. It is not used by the U14C since wait states are not required.

3-1-9: I/O CH CK

This signal is used in conjunction with data parity checking. It is not used since parity is not checked by the U14C.

A Table of "Pin-Out's" for the 8 Bit I/O channel is shown in Table 3-2.

SECTION 3-2: ADDRESS SELECTION

The U14C is operated as an I/O mapped device. Each board occupies a block of 4 contiguous I/O addresses. The factory default addresses are from base address 300 through 303 hex. (Refer to Figure 2-1) The actual address is chosen by jumpers on J15. Connecting a jumper selects a 0 for that address bit, while no jumper selects a 1. (See Section 2-3)

Table 3-2 I/O Channel Connector Pin List

PIN	DESCRIPTION	PIN	DESCRIPTION
B1	Ground	A1	I/O CH CK
B2	Reset Dry	A2	D7
B3	+5 Volt	A3	D6
B4	IRQ2	A4	D5
B5	-5 Volt	A5	D4
B6	DRQ2	A6	D3
B7	-12 Volt	A7	D2
B8	Unused	A8	D1
B9	+ 12 Volt	A9	D0
B10	Ground	A10	IO CH RDY
B11	MEMW*	A11	AEN
B12	MEMR*	A12	A19
B13	IOW*	A13	A18
B14	IOR*	A14	A17
B15	DACK3	A15	A16
B16	DRQ3	A16	A15
B17	DACK1	A17	A14
B18	DRQ1	A18	A13
B19	DACK0	A19	A12
B20	CLK	A20	A11
B21	IRQ7	A21	A10
B22	IRQ6	A22	A9
B23	IRQ5	A23	A8
B24	IRQ4	A24	A7
B25	IRQ3	A25	A6
B26	DACK2	A26	A5
B27	T/C	A27	A4
B28	ALE	A28	A3
B29	+5 Volts	A29	A2
B30	OCS	A30	A1
B31	Ground	A31	A0

* Indicates low true signal

SECTION 3-3: USING INTERRUPTS

Full interrupt capability is provided by the U14C in accordance with the I/O channel specification. Interrupts for input buffer full (IBF), transmit buffer empty (TBE), over-travel fault and operation complete are provided. Interrupt levels 2 through 7 are jumper selectable at J17. Polled operation is also supported with separate status bits for each of the above sources. Table 3-3 shows the detail of the level select jumpers on J17.

NOTE: The "C demonstration program with interrupts" illustrated in Section 9-5 provides a good example of interrupt use.

Table 3-3 Interrupt Level and DMA Selection Jumpers

DESCRIPTION	PIN	PIN	DESCRIPTION
Local DACK	1	20	DACK3
Local DACK	2	19	DACK1
Local DRQ	3	18	DRQ3
Local DRQ	4	17	DRQ1
Local IRQ	5	16	IRQ7
Local IRQ	6	15	IRQ6
Local IRQ	7	14	IRQ5
Local IRQ	8	13	IRQ4
Local IRQ	9	12	IRQ3
Local IRQ	10	11	IRQ2

SECTION 3-4: DMA

Full DMA operation is supported for either reading or writing to the U14C on levels 1 or 3. The preceding table shows the detail of the level select jumpers (J17). Refer to Host Software Section for software details. For an example of a controlling Unidex 14 under DMA control refer to the example program "U14_CDMA.C" on the software Support Disk provided).

SECTION 3-5: I/O REGISTERS

The U14C Control Board occupies 4 contiguous addresses in PC/XT/AT I/O space. The registers associated with each address are described in the following table.

Table 3-4 I/O Register Description

ADDRESS OFFSET	FACTORY DEFAULT	DESCRIPTION
0	300 hex	Data register
1	301 hex	Done flag register
2	302 hex	Control register
3	303 hex	Status register

3-5-1: DATA REGISTER

The data register is the data communication port between the U14C and the PC/XT/AT or compatible host. All data is passed between the host processor and the U14C 68000 processor through this port. This port is full duplex and double buffered in both directions allowing data to be written to the port before the previous byte has been read and processed. This allows for faster processing of the data between the two microprocessors.

3-5-2: DONE FLAG REGISTER

This is a read only register from the U14C. The status bit indicating the done status of each axis is written by the 68000 on the U14C. The host can then read it at any time to determine the status of the motion process. It is cleared by reading the register. The bit definition is shown in Table 3-5.

Table 3-5 Done Flag Register Description

DONE STATUS REGISTER DESCRIPTION	
BIT	DESCRIPTION
0	Done status of X axis
1	Done status of Y axis
2	Done status of Z axis
3	Done status of T axis
4	(Undefined)
5	(Undefined)
6	(Undefined)
7	(Undefined)

3-5-3: INTERRUPT AND DMA CONTROL REGISTER

This read/write register allows different interrupt sources from the U14C to be individually enabled or disabled. This may be performed at any time by a write to this register. It also allows DMA requests to be generated when the IBF or TBE flags are set and enabled. DMA can be requested for only one data transfer direction at a time. The register may be read back to verify or determine the state of the interrupts.

Table 3-6 Control Register Description

BIT	NAME	CONTROL DESCRIPTION
7	IRQ_E	Interrupt enable bit. This bit must be on to enable any interrupts.
6	TBE_E	Transmit buffer empty interrupt enable bit. This bit should be checked before writing to the data register to avoid sending a character when the interrupt has been disabled.
5	IBF_E	Input buffer full interrupt enable bit
4	DON_E	Done or error status interrupt enable bit
3	Unused	
2	Unused	
0	DMA_DIR	DMA direction for requests. A 0 indicates DMA will be requested for transfers from the host to the U14C board and a 1 indicates transfers from the U14C board to the host.
1	DMA_E	Enable DMA requests. A 1 will enable DMA requests when the appropriate data transfer is requesting service.

3-5-4: STATUS REGISTER

The status register is a read only register that provides status information to the host CPU. This status is independent of the enable status of the interrupt, allowing the board to operate in a polled mode if desired.

In order to resolve the source of a done or error interrupt, the DON_S bit (bit 4) should be read first. This bit in the status register is automatically reset upon reading the status register. If DON_S flag is true, the error bits should then be read to determine if the interrupt was caused by an error condition. If no error condition is present then the done flag register can be read to determine which axes are done. An "IC" command (or control-Y is preferred) should then be given from within the interrupt service

routine or poll routine to reset the done flag register and error bits. The TBE_S bit is reset by writing to the data register and the IBF_S bit is reset by reading the data register. Both bits may remain true after a single read or write since the data port is double buffered and may still remain ready for data.

SECTION 3-6: POWER SUPPLY REQUIREMENTS

The U14C is designed to operate from the power supplied in the PC/XT/AT backplane. The host computer or expansion box must be capable of supplying 2.5 amps maximum for operation of the U14C board.

Table 3-7 Status Register Description

BIT	NAME	CONTROL DESCRIPTION
7	IRQ_S	Interrupt request status.
6	TBE_S	Transmit buffer empty status. This high true bit indicates a character may be written to the input buffer of U14.
5	IBF_S	Input buffer full status. This high true bit indicates a character is available in the output buffer of U14.
4	DON_S	Done or error status. This high true bit indicates the command is complete i.e. and ID command has been executed or an error has been detected. If bits 0-3 are all false it indicates a command completion i.e. an ID command has been executed. The error bits indicate one or more errors have been detected.
3	OVRT	Overtravel. An overtravel switch was true indicating attempted travel out of bounds.
2	ENC_S	Encoder error. This bit flags a slip error on models with the encoder option if the interrupt on slip (IS) command has been issued.
1	INIT	Init flag. This bit indicates the U14C is being reset or the 68000 microprocessor has not completed initialization. Host initialization routines should check this bit for a zero before proceeding.
0	CMD_S	Command error. An unrecognizable command has been detected or LS or LE commands are not in matched pairs.

CHAPTER 4: CHASSIS, I/O AND DRIVER INTERFACE

SECTION 4-1: INTRODUCTION

This chapter describes the various Unidex 14 Chassis interface connections.

The drive modules available that interface to the Unidex 14 are:

Stepping Motor Drive Modules:

- D1401
- DM1501
- DM4005
- DMV8008
- D3001
- DM4001
- DM6006
- DMV16008

DC Motor Drive Modules

- DSL8020
- DSL3015
- DSL16020
- DSL4020

Aerodrive

- DAV4008
- DAV16008

SECTION 4-2: UNIDEX 14 REAR PANEL CONNECTIONS

An outline drawing for the rear panel of the Unidex 14 Driver Chassis is shown in Figure 4-1A and 4-1B.

Control and Power connections for both chassis shown in Figure 4-1. A brief description of the various connectors is outlined below:

XJ1,YJ1,ZJ1,TJ1

Power connection motor connectors.

J13,J14,J19*,J20*

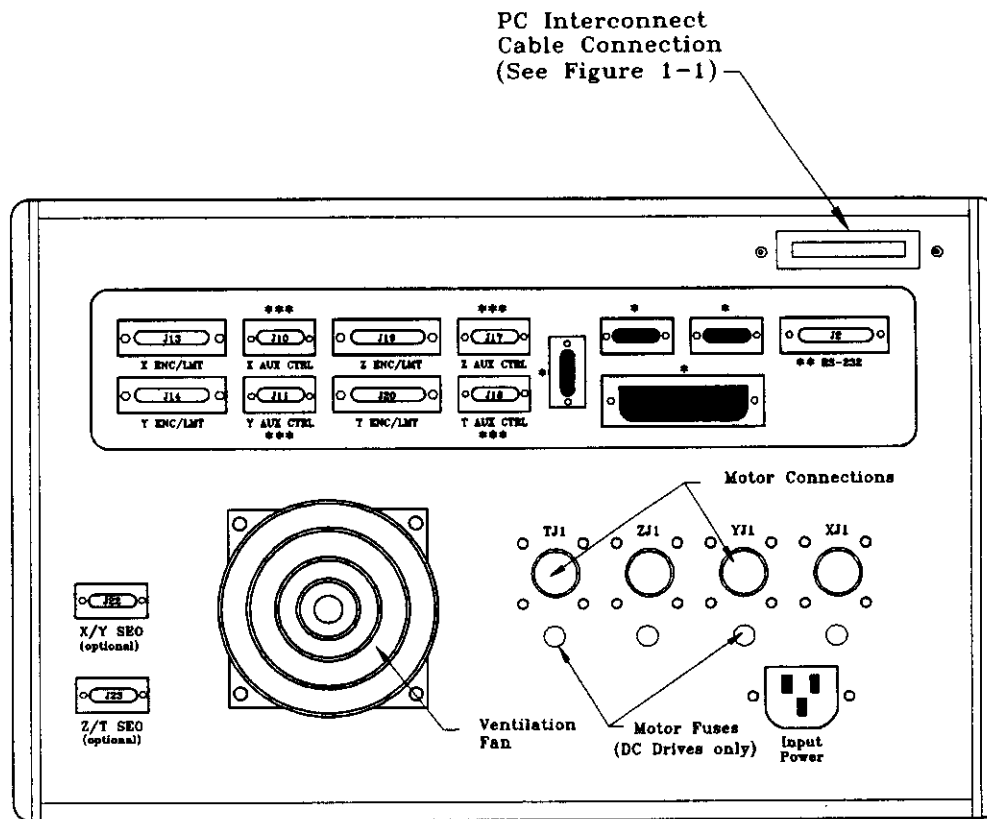
Limit, Marker, and, for the DC Servo Control, Encoder interface connectors for the X,Y,Z and T drives.

J10,J11,J17*,J18*

Auxiliary control connections for the X,Y,Z and T drives. The use of these connectors depends on the type of Unidex 14 chassis used. See Section 4-3.

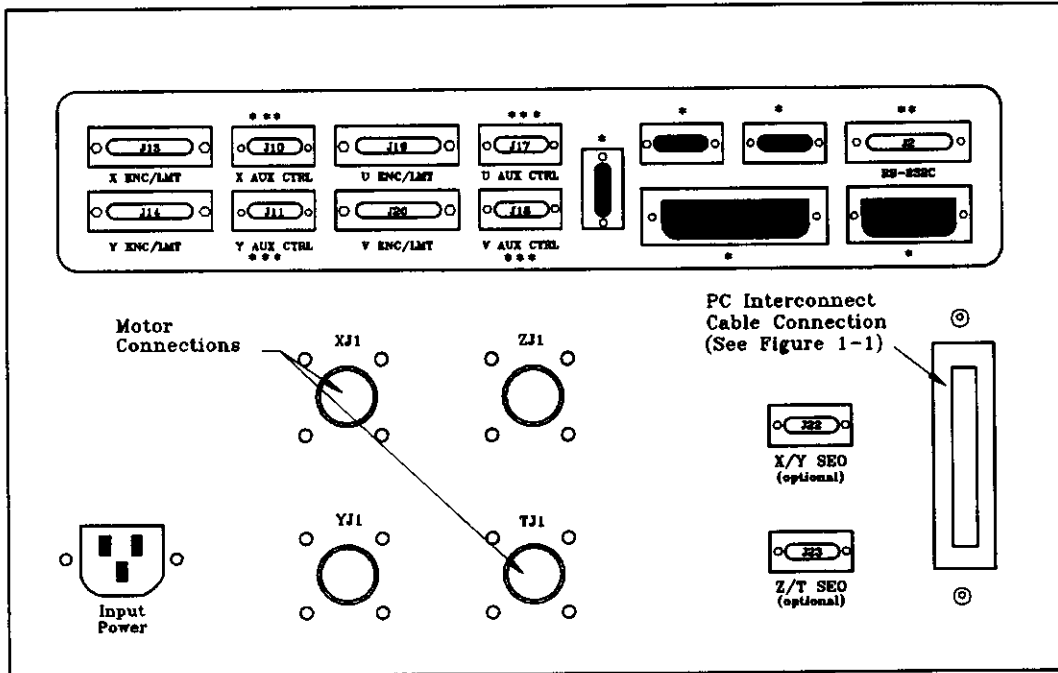
** Application for 4 Axis chassis, U14R, U14S and U14H*

J3	N/A
J5	N/A
J2	I/O
J15	N/A
J6	N/A
J25	N/A
J22,J23	These connectors are supplied for the Serial Output (SEO) Option for Unidex 14. These optional connectors provide opto-coupled positioning data and status output signals relative to X,Y,Z and T axes.



- * NOTE: Connectors are present but not used.
- ** NOTE: J2 is labeled as RS-232. The Unidex 14 however, uses this connector as an I/O connection.
- *** NOTE: These connections are used for linear encoder interface and Aux. control.

Figure 4-1A: Rear Panel Outline of Unidex 14 Four Axis Desktop & 19" Rack Chassis (Models U14S & U14R)



- * NOTE: Connectors are present but not used.
- ** NOTE: J2 is labeled as RS-232. The Unidex 14 however, uses this connector as an I/O connection.
- *** NOTE: These connections used for linear encoder interface.

Figure 4-1B: Rear Panel Outline of the Unidex 14 Four Axis Heavy-Duty 19" Rack Chassis (Model U14H)

4-2-1: UNIDEX 14 CONTROL CONNECTIONS

The external control connectors shown in Figure 4-1 are detailed in Figure 4-2. Aerotech supplies the mating connectors for each external control connector upon request.

For convenience, a list of mating connectors and their manufacturer is presented in Table 4-1:

Mating connector for 25 pin "female" "D" connectors J13,J14,J19 and J20	Molded cable type "male", Beldon, No. 49670 Solder pot type "male", TRW-CINCH, No. DBM-25P Ribbon connector type "male", TRW-CINCH, No. FC25P
Mating connector for 25 pin "male" "D" connector J2	Molded cable type "female", Beldon, No. 49675 Solder pot type "female", TRW-CINCH, No. DBM-25S Ribbon connector type "female" TRW-CINCH, No. FC25S
Mating connector for 15 pin "male" "D" connectors J10,J11,J17,J18,J22,J23	Molded cable type "female", Beldon, No. 49727 Solder pot type "female", TRW-CINCH, No. DAM-15S Ribbon connector type "female", TRW-CINCH, No. FC15S

Table 4-1: List of Mating Connectors for External Control Connectors

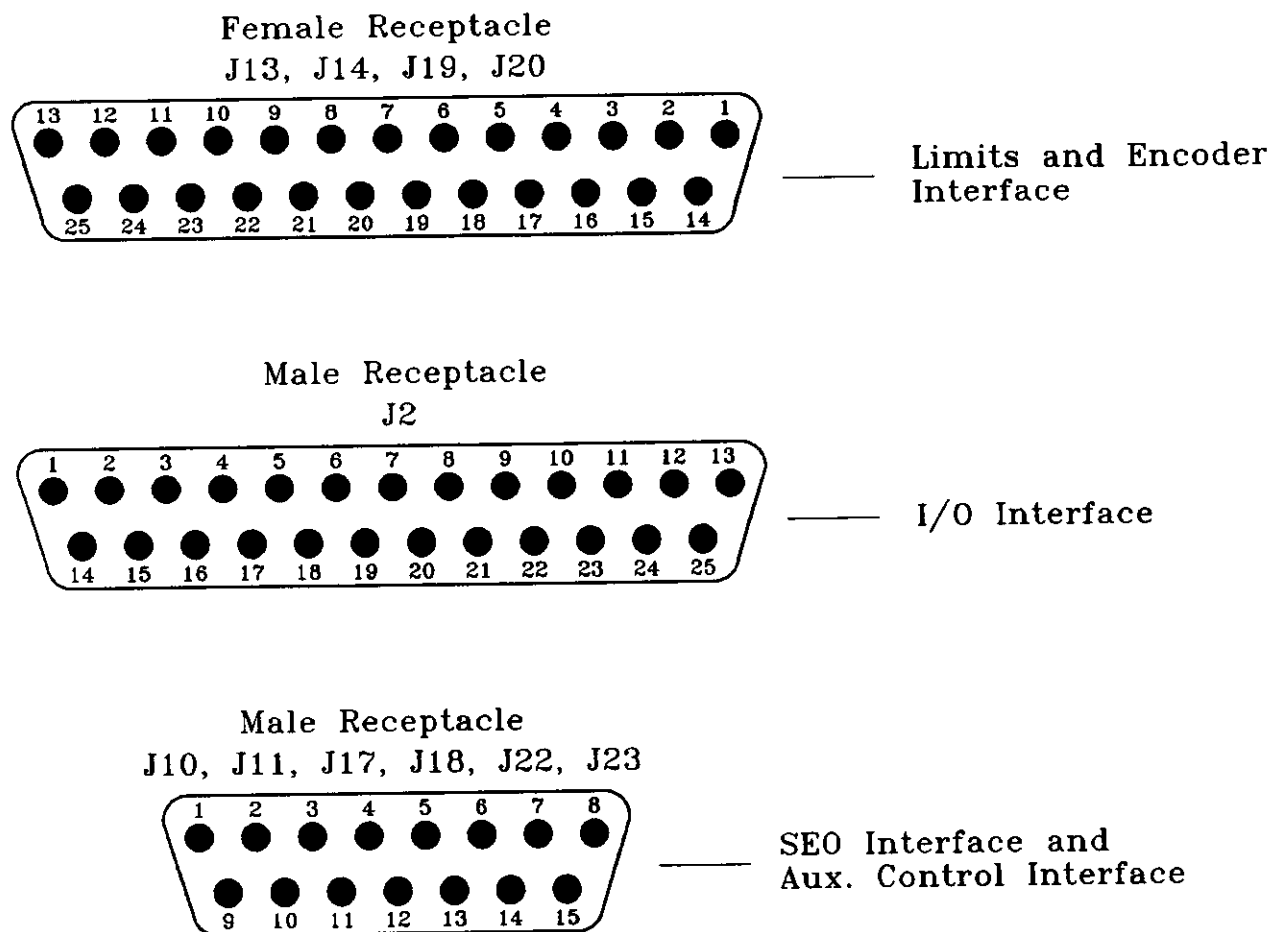


Figure 4-2: Outline of Unidex 14 External Control Connectors

4-2-2: LIMIT SWITCH, MARKER AND ENCODER

Connections for X,Y,Z,T Drives

The following section describes the external control connections of the Unidex 14 (See Figures 4-1A, 4-1B and 4-2 for more detail).

4-2-2-1: CONNECTOR SPECIFICATION

J13,J14,J19 and J20 for U14S,U14R and U14H Drive Chassis

Connectors J13, J14, J19 and J20 provide for the termination of the basic control interface signals between the X,Y,Z and T Aerodrive, Stepping Motor and/or DC Servo Motor and the Unidex 14. Motor travel limits and homing signals for each axis are terminated at these receptacles, as well as Encoder signals for DC Motor and Aerodrive control. Encoder tracking for open loop stepping control. The pinouts are as follows (See Figure 4-2 for receptacle outline).

J13,J14,J19 and J20 (U14S,U14R and U14H Chassis):

<u>Pin #</u>	<u>Description</u>	<u>Comments</u>
	FORM C	FORM A (normally open) or FORM B (normally closed)
13 *	CW Limit (in)	No Connection
12 *	/CW Limit (in)	/CW or CW Limit (in)
25 *	/CCW Limit (in)	No Connection
24*	/CCW Limit (in)	/CCW or CCW Limit (in)
23 *	Home Limit (in)	No Connection
22 *	/Home Limit (in)	/Home or Home Limit (in)
7	Marker (in)	Encoder Marker (or Index Pulse, see following diagram) for DC Servo Motor or separate marker (Index Pulse for Stepping Motor control).
6	/Marker (in)	
3,16	+ 5VDC (200mA max)	For Encoder only.
9,20,21	Signal Common	(No connection)
4	Optional +5VDC (For Aerotech use only)	
1	Shield (Connector Retaining Screws are also connected to chassis frame)	
14	COS (in)	Encoder quadrature signals (see following diagram).
15	/COS (in)	
17	/SIN (in)	
18	SIN (in)	
19	+ Tach Input	Optional Tach for DC Servo only Polarity indicated for CW Motor rotation
8	- Tach Input	

* Form C Limit connections are "factor select" only. Form A Limit connections are the standard configuration shipped. Form B Limit connections can be field selected on the given DC or Stepping Drive module.

Encoder Input Specifications:

The Encoder feedback option is intended primarily for applications where desired positional accuracy exceeds the accuracy of the mechanical drive components (for example; lead screws) or position feedback is required to detect motor slip or stall. The Encoder Interface is required for DC Brush and Aerodrive Systems. It is optional for open loop stepping systems.

NOTE: Encoder position verification commands (See Sections 8-13, 8-14, 8-15 and 8-16) may optionally be used for DC Brush and Aerodrives. The DC Brush and Aerodrive Systems use the encoder internally to close the servo (positions) loop.

Unidex 14 accepts quadrature pulse outputs from high resolution encoders. Up to 50,000 pulse per revolution encoders may be used while the indexers are generating pulses at their maximum rate. This allows position feedback information to match the resolution to the microstepping motor drive.

Times four quadrature detection is used to increase resolution. The inputs are compatible with encoders which have single ended TTL outputs as well as differential line drivers. Provisions are also provided for an index pulse (differential or single ended).

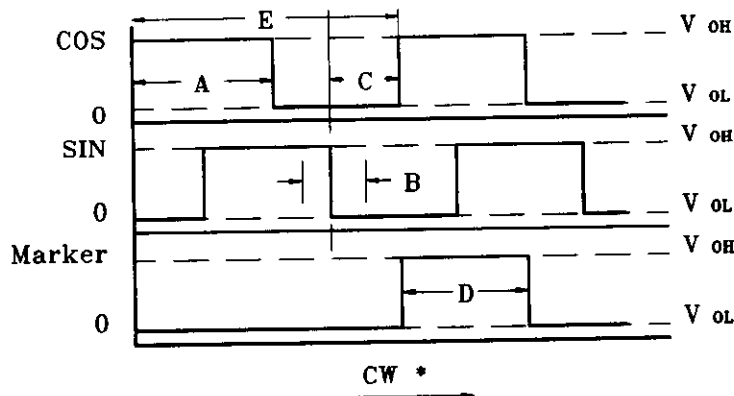
The user can specify the encoder count/motor count ratio for position maintenance and encoder tracking mode. This ratio is handled internally in floating point format and can be virtually any ratio. Slip detection requires that the encoder resolution (after the 4X quadrature detection) match the motor resolution.

The Unidex 14 DSL8020, DSL3015, DSL4020 or DSL16020 DC Servo Modules can be factory set for:

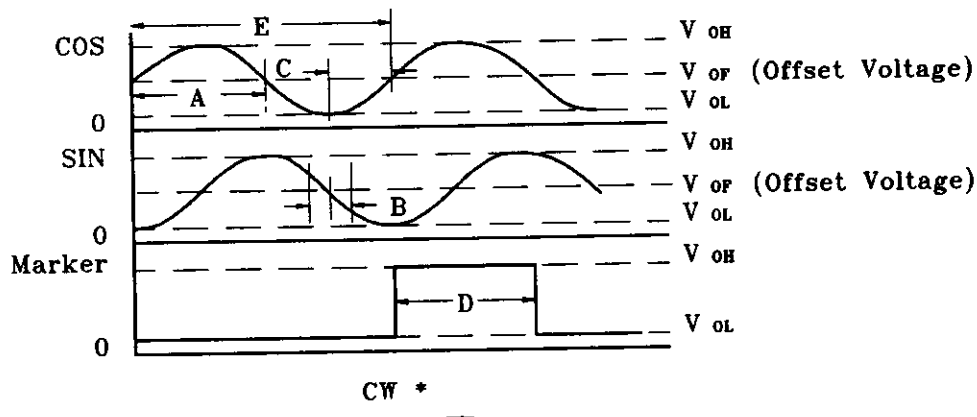
1. Differential amplified sine wave encoder operation, or
2. Single-ended or differential square wave (line driver or TTL) encoder operation.

Electrical specifications for both sine and square wave Encoder types are shown below:

Square Wave Encoder
(/COS, /SIN, and /Marker not shown, but similar)



Sin Wave Encoder
(/COS, /SIN, and /Marker not shown, but similar)



* Motor rotational reference, CW, is looking into the motor mounting flange. Encoder is assumed to be mounted on rear of motor.

NOTE: DC Brush Systems and Encoder Verification Systems can use both sin and square wave forms. However, Aerodrive Systems can only use sin wave encoders.

Encoder Voltage Specifications:

<i>Specification</i>	<i>Sine Wave</i>	<i>Square Wave</i>
V OH	3.0 V MIN	2.4 V MIN
V OL	1 V MAX	.4 V MAX
V OFF	1.75 - 2.25 V TYP	-----

Encoder Timing Specifications:

<i>Specification</i>	<i>Sine Wave</i>	<i>Square Wave</i>
A(Symmetry)	180°±10°Max	180°±10°Max
B(Jitter) (1)	±20°Max	±5°Max
C(Quadrature) (1)	90°±20°Max	90°±20°Max
D(Marker Width) (2)	90°Min/270°Max	90°Min/270°Max
D(Period) (3)	25mSec Min (40KHz)	13.3mSec Min (75KHz)

NOTE 1: Quadrature jitter at 35 KHz.

NOTE 2: The DC Servo Drive Modules internally mask the Marker to "one" machine step, based on the logical "and" of the "positive" Sine and Cos single states.

NOTE 3: The maximum Encoder frequency stated above is based on one full output cycle of the Encoder (X1 mode). The DC Servo Drive Modules can be field selected for X1, X2, or X4 interpolation of the Encoder signals. The maximum data rate of the Unidex 14 system is 524kHz. When a sine wave Encoder is used in the X4 mode, the maximum system data rate is 160kHz. When a square wave Encoder is used in the X4 mode, the maximum system data rate would be 300kHz.

Single-ended square wave Encoders can be accommodated by appropriately connecting the Encoder to the Sin, Cos and Marker connections, while leaving the /Sin, /Cos and /Marker points unconnected.
(NOTE: For "noisy" environments, differential output Encoders are recommended.)

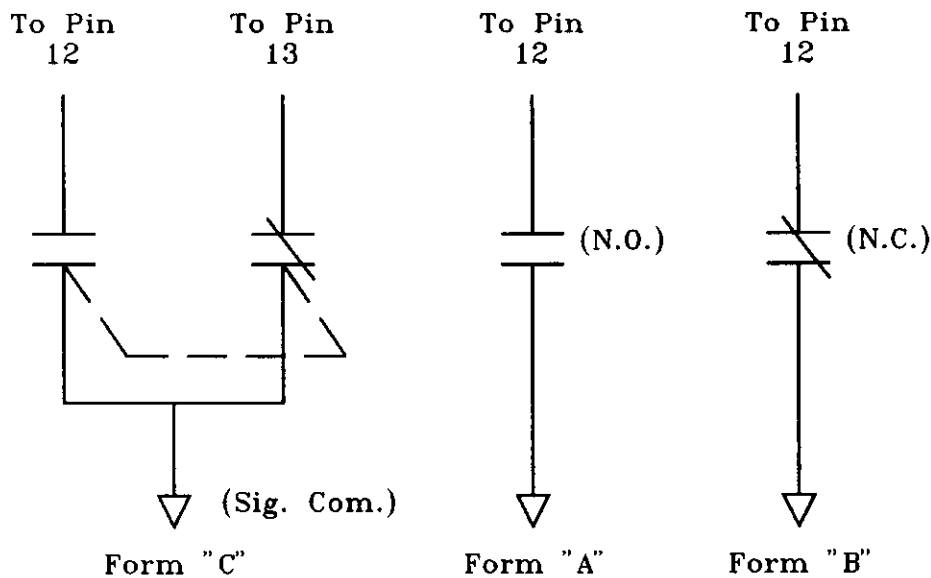
The Marker inputs connections are applicable for Aerodrive as well as DC Servo Motor control. However, the electrical specifications for the Marker inputs (pin 7 and 6) when used with Stepping Motor control without the optional Tracking Encoder, differ from the Encoder Marker specifications just previewed for DC Servo control.

The Stepping Motor Drive Systems (ie., Drive Modules DM4005, DM6006 etc.) may be equipped with a Home Marker option (option number "HMO") that is not associated with the marker pulse of an encoder. The Internal Interface circuit for this option is shown in Section 4-2-2-2-1.

4-2-2-2: SPECIFICATIONS FOR SECTION 4-2-2-1

Form C connections for the CW, CCW and Home Limit inputs are optional. Form A and form B connections are standard, and are connected at pins 12, 24 and 22 respectively.

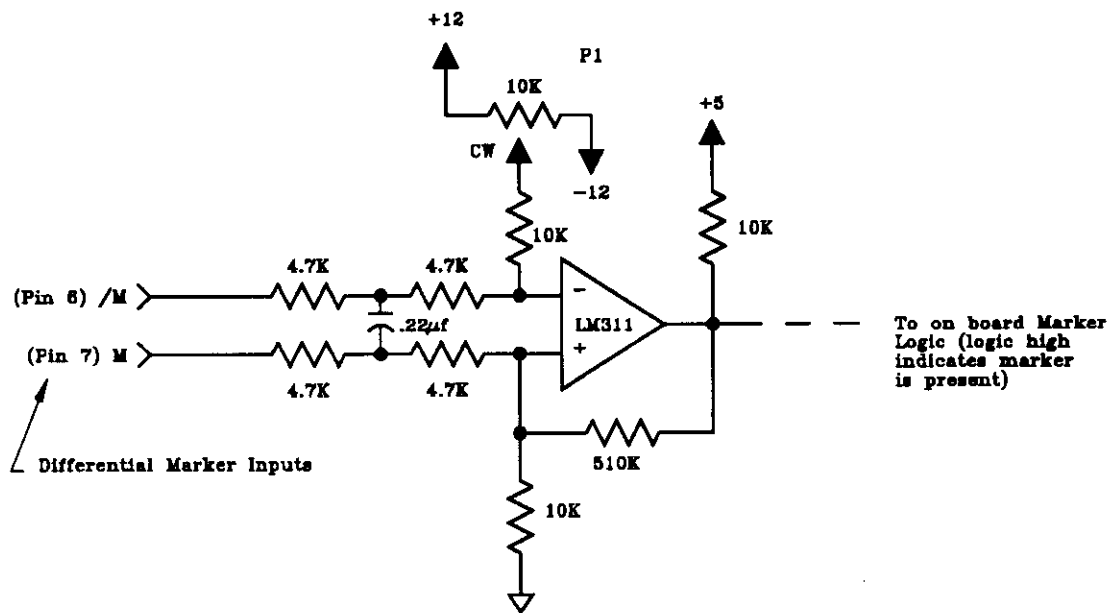
Form C, form A and form B contacts are defined for the CW limit as follows (CCW and Home limits are similar).



Contact Positions shown when not in limit.

4-2-2-2-1: HOME MARKER INTERFACE SPECIFICATIONS FOR "DM" SERIES STEPPING MOTOR DRIVE

Shown below is a circuit diagram of the differential input marker circuit, for open loop stepping drives using the Home Marker options, ("HMO"). This circuit is located on the "DM" Series Stepping Drives.



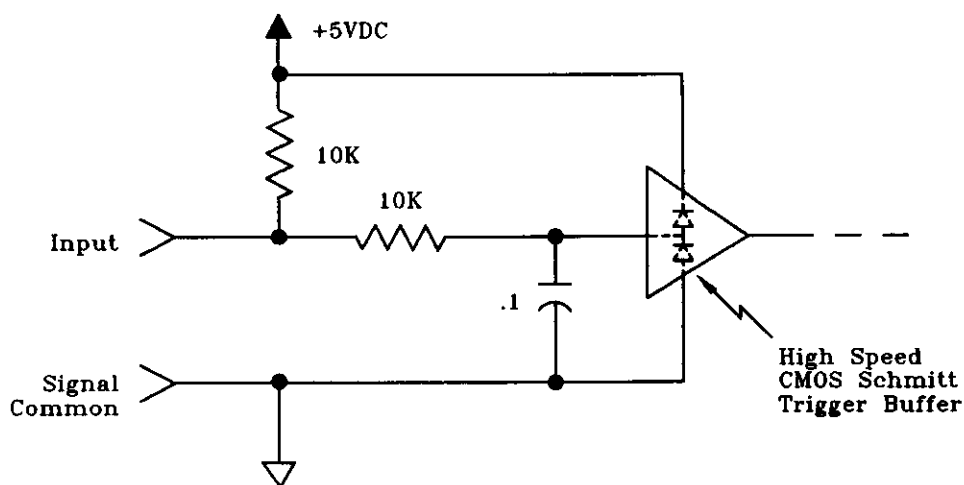
Potentiometer P1 in the circuit above, allows the threshold of the marker input signal to be adjusted. The threshold level is increased by turning P1 CW. To activate the marker logic, the voltage signal at "M" must rise slightly higher than the voltage signal at "/M", plus the threshold setting at P1. In other words:

$$M > /M + P1 \text{ (threshold)}$$

When using form A or form B input connections, selections of polarity (normally open or normally closed) must be made on the appropriate module through a set of programming jumpers. (See Chapter 6 for more information)

Normally, the polarity of these limits are factory set for Form A (normally open).

All limit input configurations are internally buffered as shown below:



All limit inputs are protected against accidental over voltage of ± 30 volts. All logic inputs such as dry contact, open collector, TTL, 5 to 15 volt CMOS can be used.

The maximum current draw (per axis) on the +5VDC connections (pins 3 and 16) is 200 mA.

The presence of a visual indication for acknowledging CW limit, CCW limit and Marker can be made by viewing the LED indicators on the front panel for a given axis.

The Drive Module specifications in Chapter 6 of this manual provide more information concerning the Limit switches, the Home cycle, and Marker references.

The Limit and Home switches are contact closures or any other suitable active switch such as a hall effect switch or opto-isolator to ground. The Limit switch closure will stop the associated pulse stream if the motor travels beyond its allowable limits and trips the switch. The Home switch Limit provides a means to synchronize the motor controller with the load at some home or reference position. The Home Limit switch when used with the software HM (See Section 8-4) command will cause the motor to stop when the switch closes. On finding the home position the internal position counters will be initialized. The sense of the home switches may be changed to true when open if desired by use of the HH command. The limit switches may be changed to true when open by removing the jumper on J47 of the U14C Control Board if desired.

The Unidex 14 provides "Homing" to an initial reference in one of two ways:

First, by the utilization of the software commands (listed in Sections 8-3 and 8-9), the Unidex 14 Control Board may be set to "Home" either to the CW or CCW switches. This "Homing" method provides the versatility of "Homing" speed adjustment as well as the ability to send an individual axis to home without effecting the position of the other axis.

The second method of "Homing" utilizes the Hardware Home logic of the individual Driver Modules. (See Chapter 6) This is the preferred method of "Homing" since the home position is referenced to the Index or Marker power of an encoder or similar device

NOTE: The Hardware Home feature utilizes Input 5 and Output 13 of the J2-I/O connector as control signals to the Unidex 14 Control Board. Thus, if Hardware Homing is selected, pins 5 and 13 of the J2 connector cannot be used as general purpose I/O (See Chapter 6 for more information.)

SECTION 4-3 AUXILIARY CONTROL CONNECTIONS

4-3-1: CONNECTOR SPECIFICATIONS

For U14S, U14R, and U14H Chassis:

J10, J11, J17 and J18 for DSL and DAV Modules (DC Brush and Aerodrive Modules)

The auxiliary control connections for the U14S, U14R and U14H chassis are associated with connectors J10, J11, J17 and J18 for the X, Y, Z and T axes respectively. For DC Servo and Aerodrive operation the pin definitions for J10, J11, J17 and J18 are as follows: (See also Figures 4-1A, 4-1B and 4-2.)

<u>Pin#</u>	<u>Description</u>	<u>Function</u>
13	External clock (in) Max Frequency = 250KHz Min. Pulse Width = .5 m S	Remote input clock and direction control (Used if modules are enabled for remote mode, see Section 4-4)
6	External Direction (in) Logic 1 = CW Motor rotation. Logic 0 = CW Motor rotation.	
14	CWFB (out)	Buffered Encoder feedback output clock signal for external synchronization applications. Output pulse width per machine step = 1 μ Sec (Not Opto-Isolated)
7	CCWFB (out)	
15	/Reset (out)	Axis reset status output (Driver low for "Loss of Encoder" and current trip).
8	/Marker (out)	Buffered Output Signal indicates state of Marker Input signal (See Section 4-2-2).
9	COS (in)	Encoder input signal. (Aerotech uses these inputs for linear encoder interfacing applications leaving limit connections on J13,J14,J19,J20 fully accessible to user.) (See Section 4-2-2)
2	/COS (in)	
10	SIN (in)	
3	/SIN (in)	
11	Marker (in)	+ 5VDC for linear encoder use only.
4	/Marker (in)	
12	+ 5VDC (200 mA max)	
5	Signal common	
1	Shield	

4-3-2: CONNECTOR SPECIFICATIONS

U14S,U14R and U14H Chassis

J10,J11,J17 and J18 for D, DM and DMV Modules (Open Loop Stepping Modules)

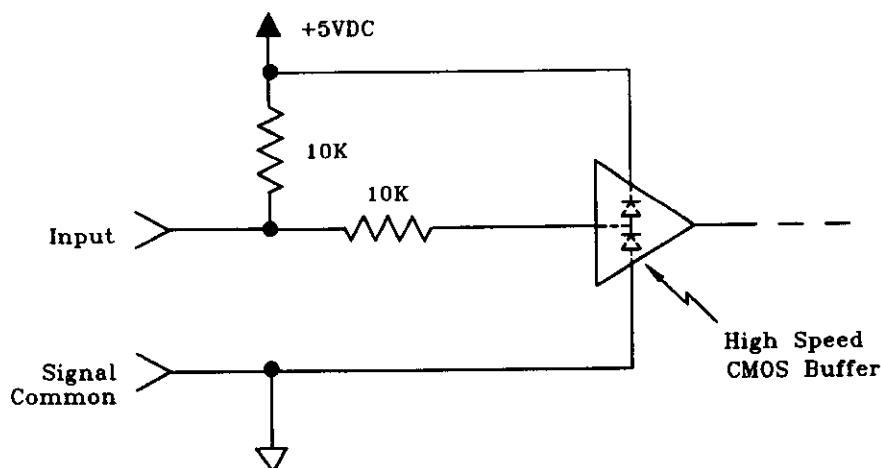
The auxiliary control connections for the U14S, U14R and U14H Chassis assume different pin-out definitions when used with the low power Stepping Motor Drive Modules (D1401,D3001,DM1501,DM4001,DM4005,DM6006) or high power Stepping Motor Drive Modules (DMV8008,DMV16008) in the open loop configuration without Encoders. The pin definitions for these connectors when using the D,DM or DMV Series Stepping Modules are as follows:

D, DM or DMV Series-J10,J11,J17 & 18 (U14S,U14R and U14H Chassis)

<u>Pin#</u>	<u>Description</u>	<u>Function</u>
13	External clock (in), Max Frequency = 250KHz. Min. Pulse Width = $.5\mu\text{S}$	Remote Input Clock and direction control (used if modules are enabled for Remote mode, see Section 4-4)
6	External direction (in) Logic 1 = CW Motor Rotation Logic 0 = CCW Motor Rotation	
2-4,7-12,14,15		(Connection for Aerotech use only)
5	Signal common	
1	Shield	

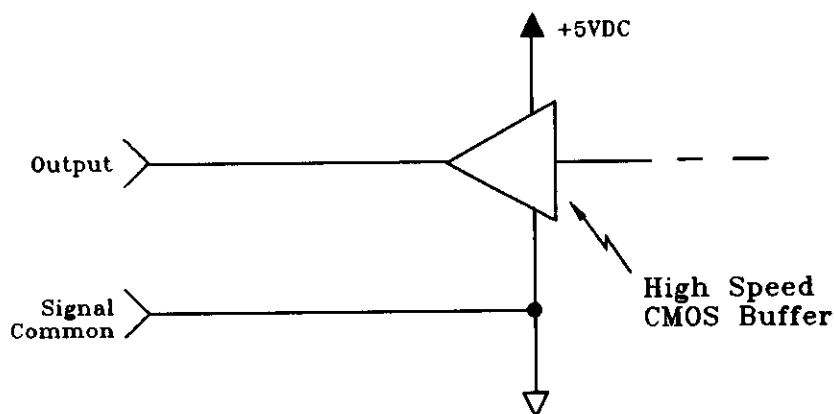
4-3-3: AUXILIARY CONTROL ELECTRICAL SPECIFICATIONS

The circuit configuration below illustrates input buffering for pins 13 and 6 described in Sections 4-3-1 and 4-3-2, and buffering for pins 12, 13, 22, 23, 24 and 25 of Section 4-2-2-1.



These inputs are protected against accidental overvoltage of ± 30 volts. These inputs will accept dry contact, open collector, TTL and 5 to 15 volts CMOS.

The circuit configuration below illustrates output buffering for pins 14, 7, 15 and 8 described in Section 4-3-1.



The specifications for these outputs are as follows:

Maximum current sourcing	+ 20mA
Maximum current sinking	- 20mA
Unloaded logic levels	+ 5 VDC logic high 0 VDC logic low
Loaded logic levels (@ ± 5 mA sourcing or sinking)	+ 4.8 VDC logic high + .18 VDC logic low

SECTION 4-4: OPTO ISOLATED I/O INTERFACE CONNECTOR

Logic level input/output (I/O is provided through pin connections on connector J2 (J2 may be labeled "RS-232". Ignore this label, refer to Figures 4-1A and 4-1B).

An outline of J2 is shown in Figure 4-2.

All input/output logic connections on J2 are opto-isolated. It is important to note that some of these input and output connections have auxiliary (dual purpose) uses.

A pin description of pin connections for J2 is shown in Section 4-4-1.

A diagram of the input/output connector is shown in figure 4-2. The source current for all opto-coupler inputs requires 7mA of current minimum. The output opto-couples are capable of 10mA sinking at 20 VDC.

4-4-1: J2 (I/O) CONNECTOR SPECIFICATIONS FOR THE U14S, U14R AND U14H CHASSIS

The J2 (I/O) connector pin descriptions are as follows:

<u>Pin#</u>	<u>Main Function</u>	<u>Alternative</u> (Consult Factory)
2,14	Input Opto-Coupler Anodes	None
15	Input 0	None
3	Input 1	None
16	Input 2	None
4	Input 3	None
17	Input 4	None
5	No Function	Input 5
18	Input 6 (Stepping Drives with or without 1MR Board)	None
	No Function (DC Brush Servo or Aerodrive Servo)	Input 6
6	Input 7 (Stepping Drives without 1MR Board only)	None
	No Function (Stepping Drives with 1MR Board,DC Brush Servo, Aerodrive Servo)	Input 7
20	Output 8	None
8	Output 9	None
21	Output 10	None
9	Output 11	None
22	No Function	Output 12
10	No Function	Output 13

<u>Pin#</u>	<u>Main Function</u>	<u>Alternative</u>
7, 11, 19, 23	Output Opto-Coupler Emitters	None
24	Aux. X Output (Stepping Drives with 1MR Board,DC Brush Servo, Aerodrive Servo)	None
	No Function (Stepping Drives without 1MR Board only)	Aux. X Output
12	Aux. Y Output (Stepping Drives with 1MR Board,DC Brush Servo, Aerodrive Servo)	None
	No Function (Stepping Drives without 1MR Board only)	Aux. Y Output
25	Aux. Z Output (Stepping Drives with 1MR Board,DC Brush Servo, Aerodrive Servo)	None
	No Function (Stepping Drives without 1MR Board only)	Aux. Z Output
13	Aux. T Output (Stepping Drives with 1MR Board,DC Brush Servo, Aerodrive Servo)	None
	No Function (Stepping Drives without 1MR Board only)	Aux. T Output

SECTION 4-5: SERIAL OUTPUT (SEO) INTERFACE

The Serial Output Interface (SEO) provides opto isolated (standard) or TTL/CMOS (optional) position information for external use (such as position counters/displays and speed indicators). Each SEO board (Figure 4-3) provides outputs for two axes, so two SEO boards are required for three or four axis applications. Outputs are present on the chassis rear panel at connectors J22 (X and Y axis) and J23 (Z and T axis). See Figures 4-1A, 4-1B and 4-2 for an outline of these connectors.

The outputs present at J22 and J23 vary for Stepping Motor Drives (D,DM and DMV Series) as opposed to DC Servo and Aerodrive Motor Drives (DSL and DAV Series). In addition, certain output variations and polarity inversions are possible within each subdivision. For J22/J23 outputs when DC Servo and Aerodrive Motor Drives are being used, see Table 4-3. For J22/J23 outputs when Stepping Drives are being used, see Table 4-2. Note that each table indicates the standard output functions, as well as applicable jumper information needed to change to certain optional output functions. Figure 4-6 shows the typical circuit for each available output. (These circuits are per axes)

The output signals for Stepping Motor Drives are Clock, Direction and /Marker or /Reset. (Note that a slash (/) indicates active low outputs, and no slash indicates active high.) The output signals for DC Servo and Aerodrive Motor Drives are CW Clock, CCW Clock and /Marker or /Reset. The /Marker signal is standard for both types of drives and the /Reset signal is optional. The following is a brief definition of each output.

- | | |
|-------------------------|---|
| 1. Clock - | This output is a mirror of the axis command clock. Each pulse represents one commanded step. (Stepping Drive only) |
| 2. Direction - | This output is a mirror of the axis command direction. Logic high (+ 5VDC) indicates CW motor rotation, logic low (0VDC) indicates CCW motor rotation. (Stepping Drive only) |
| 3. /Marker (standard) - | This output mirrors the system feedback reference pulse. |
| 4. /Reset (optional) - | This output indicates axis "reset" status. |
| 5. CW Clock - | This output is a 1 μ Sec pulse that is derived from the Encoder Feedback signal when the servo motor is rotating in the CW direction. Each pulse represents one commanded machine step. (DC and Aerodrive only) |
| 6. CCW Clock - | Same as CW Clock except that it is produced by opposite direction motor rotation. (DC and Aerodrive only) |

NOTE: All cables interfacing to the SEO outputs should be limited in length to 15 ft. If longer cables are required, please consult factory for SEO modifications.

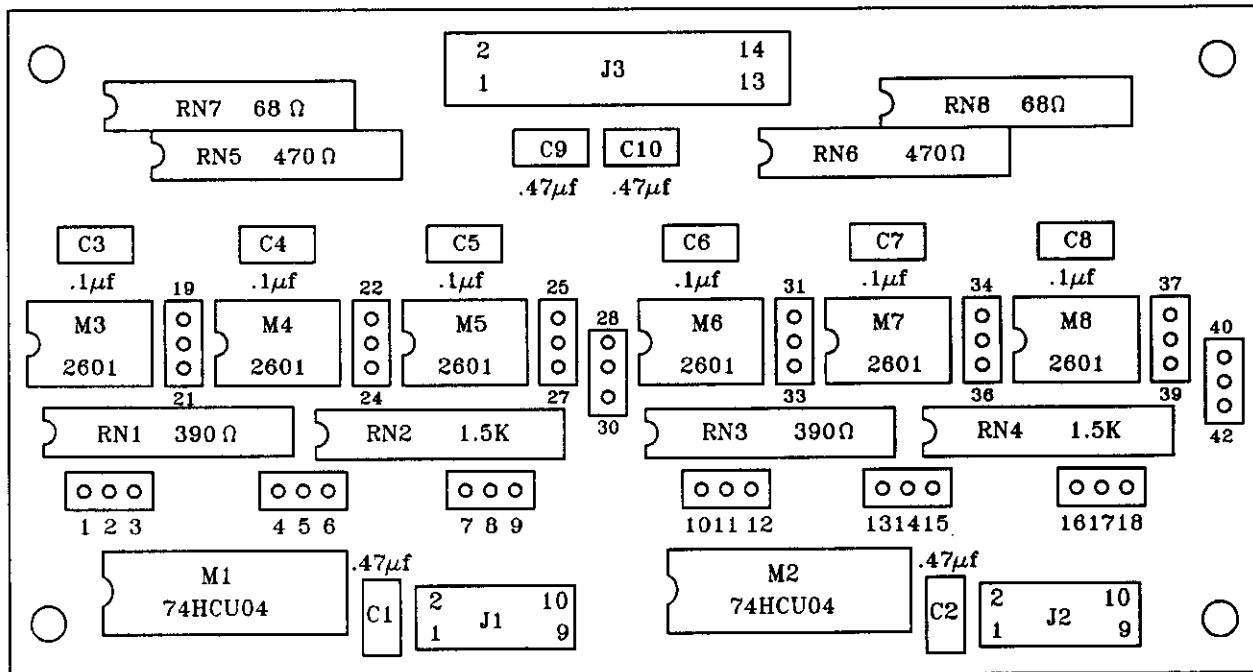
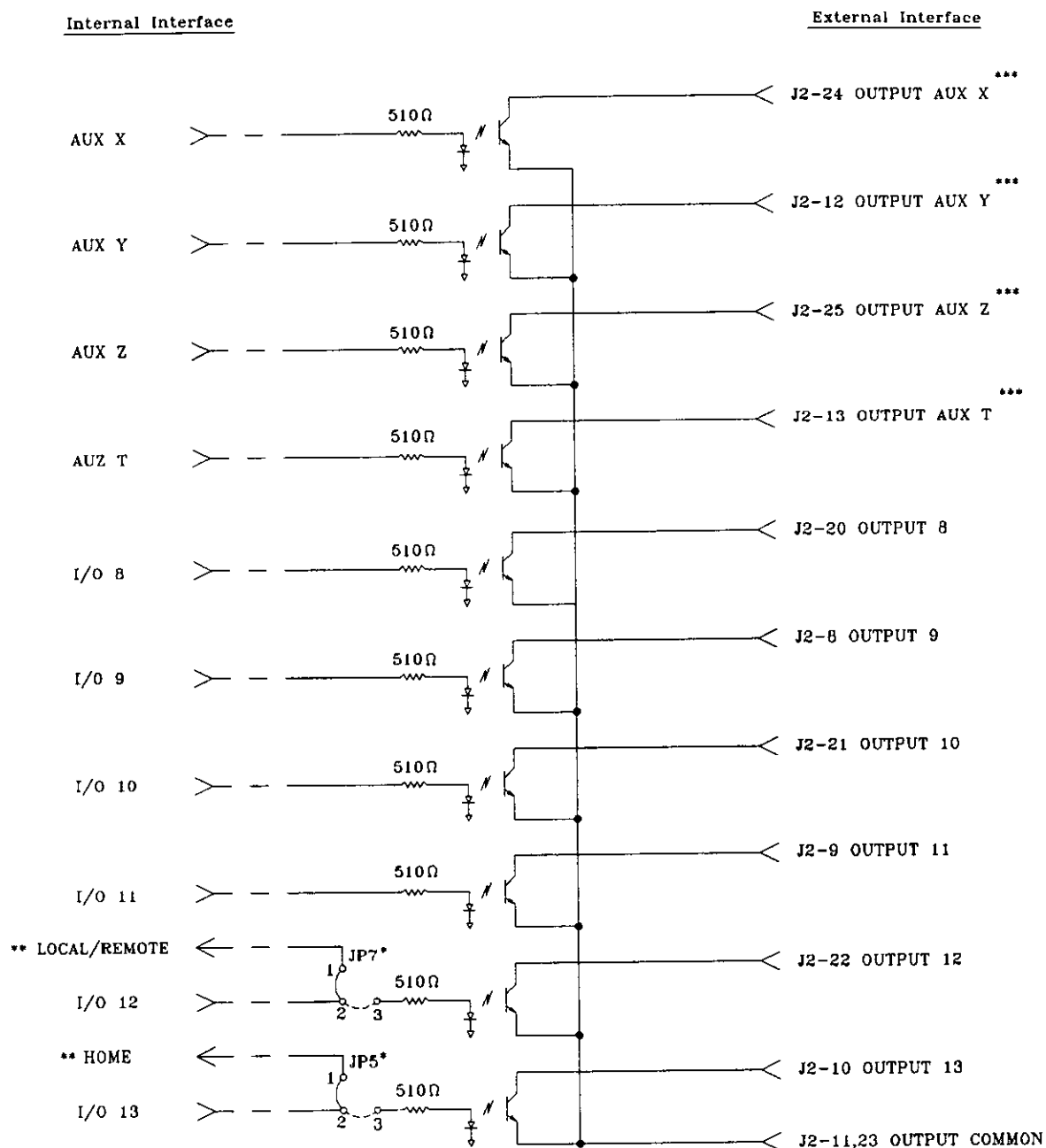


Figure 4-3: SEO Interface Board 690C1353



* Alternative Functions

** Refer to Chapter 6 for jumper locations

*** Not available on any axis having a stepping motor that is assigned for Lo/Hi current control.

Figure 4-4: External Output Connections

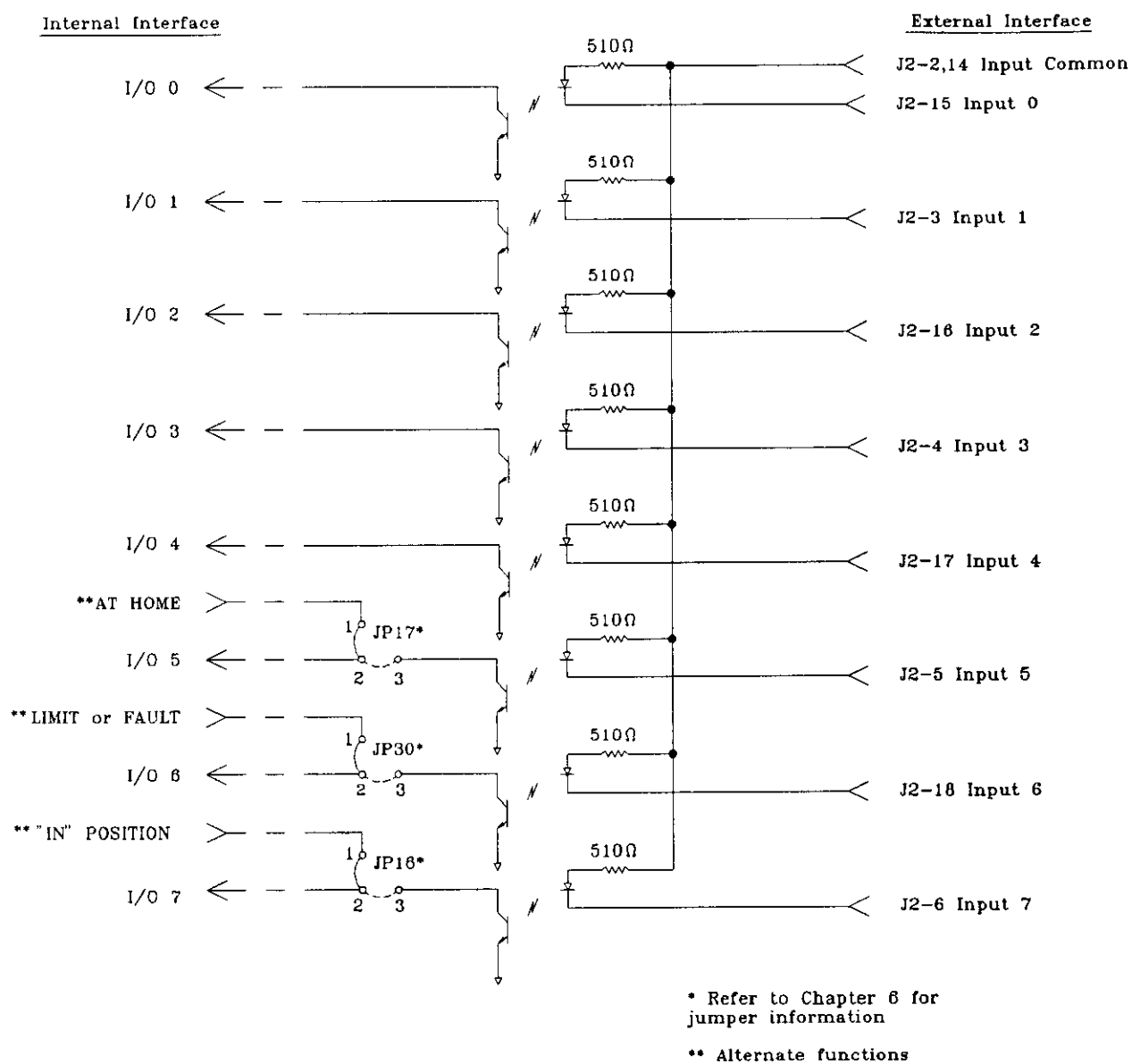


Figure 4-5: External Input Connections

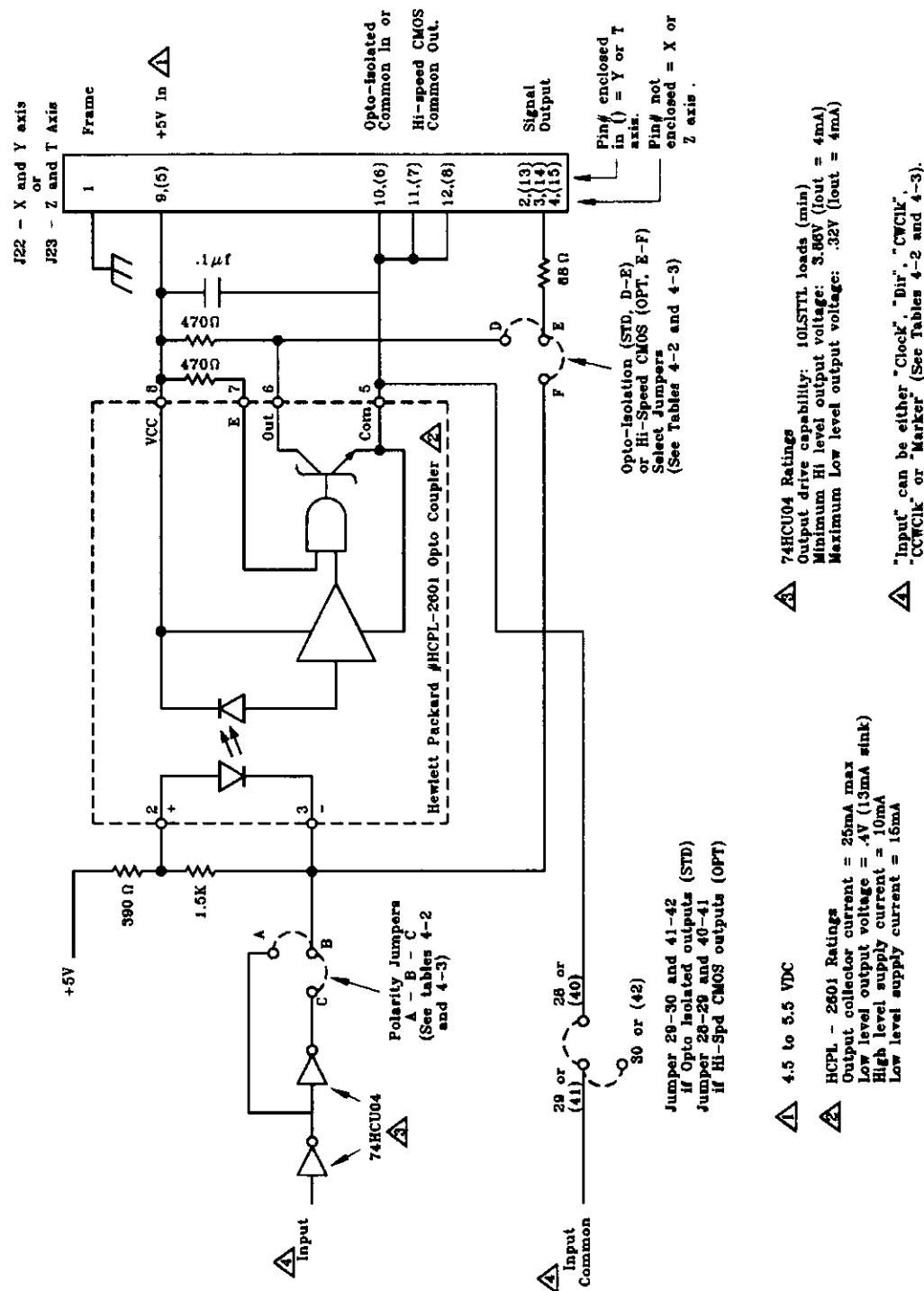


Figure 4-6: Typical "SEO" Output Circuit









J22-X,Y (J23-Z,T) Pin #	Standard Input Function for given axis	Standard Input Function, Polarity Jumper A - B - C	Optional Input Function for given axis	Optional Input Function, Polarity Jumper A - B - C	Opto Isolation Output Select (STD) Jumper  D - E - F	Hi-Speed C-MOS Output Select (Optional)  Jumper D - E - F
1	Shield		Shield			
9	X(Z)+5VDC In		X(Z)+5VDC In			
10,11,12	X(Z) Com		X(Z) Com			
2	X(Z) Clock	1-2-3	X(Z) /Clock	1-2-3	19-20-21	19-20-21
3	X(Z) Dir	4-5-6	X(Z) /Dir	4-5-6	22-23-24	22-23-24
4	X(Z)/Marker or 	7-8-9	X(Z) Marker or 	7-8-9	25-26-27	25-26-27
5	X(Z)/Reset	7-8-9	X(Z) Reset	7-8-9	25-26-27	25-26-27
6,7,8	Y(T)+5VDC In		Y(T)+5VDC In			
	Y(T) Com		Y(T) Com			
13	Y(T) Clock	10-11-12	Y(T) /Clock	10-11-12	31-32-33	31-32-33
14	Y(T) Dir	13-14-15	Y(T) /Dir	13-14-15	34-35-36	34-35-36
15	Y(T)/Marker or 	16-17-18	Y(T) Marker or 	16-17-18	37-38-39	37-38-39
	Y(T)/Reset	16-17-18	Y(T) Reset	16-17-18	37-38-39	37-38-39

Table 4-2: Stepping Motor Drive-J22, J23 Standard and Optional Input Configuration Definitions and Options

NOTES: 1 /Output = Active low (0V), Output = Active high (+5VDC).

 If opto isolators are used, add jumpers 29-30, 41-42. See Figure 4-4.

 If hi-speed CMOS, add jumpers 28-29, 40-41. See Figure 4-4.

 A third input (per axis) can be "/Marker" or "/Reset". This is **not** field changeable. Input must be configured at factory. "/Marker" input is standard unless specified otherwise.

J22-X,Y (J23-Z,T) Pin #	Standard Input Function for given axis	Standard Input Function, Polarity Jumper A - B - C	Optional Input Function for given axis	Optional Input Function, Polarity Jumper A - B - C	Opto Isolation Output Select (STD) Jumper ² D - E - F	Hi-Speed C-MOS Output Select (Optional) Jumper ³ D - E - F
1	Shield	—	Shield	—	—	—
9	X(Z)+5VDC In	—	X(Z)+5VDC In	—	—	—
10,11,12	X(Z) Com	—	X(Z) Com	—	—	—
2	X(Z) CW CLK	1-2-3	X(Z) /CW CLK	1-2-3	19-20-21	19-20-21
3	X(Z) CCW CLK	4-5-6	X(Z)/CCW CLK	4-5-6	22-23-24	22-23-24
4	X(Z)/Marker	7-8-9	X(Z) Marker	7-8-9	25-26-27	25-26-27
	or ⁴		or ⁴			
5	X(Z)/Reset	7-8-9	X(Z) Reset	7-8-9	25-26-27	25-26-27
6,7,8	Y(T)+5VDC In	—	Y(T)+5VDC In	—	—	—
	Y(T) Com	—	Y(T) Com	—	—	—
13	Y(T) CW CLK	10-11-12	Y(T) /CW CLK	10-11-12	31-32-33	31-32-33
14	Y(T)CCW CLK	13-14-15	Y(T)/CCW CLK	13-14-15	34-35-36	34-35-36
15	Y(T)/Marker	16-17-18	Y(T) Marker	16-17-18	37-38-39	37-38-39
	or ⁴		or ⁴			
	Y(T)/Reset	16-17-18	Y(T) Reset	16-17-18	37-38-39	37-38-39

NOTES:

¹ /Output = Active low (0V), Output = Active high (+5VDC).

² If opto isolators are used, add jumpers 29-30, 41-42. See Figure 4-4.

³ If hi-speed CMOS, add jumpers 28-29, 40-41. See Figure 4-4.

⁴ A third input (per axis) can be "/Marker" or "/Reset". This is not field changeable. Input must be configured at factory. "/Marker" input is standard unless specified otherwise.

Table 4-3: DC Servo and Aerodrive - J22, J23 Standard and Optional Input Configuration Definitions and Options

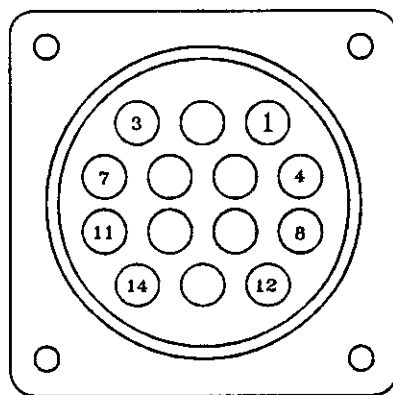
SECTION 4-6: UNIDEX 14 POWER CONNECTIONS

Input power to the Unidex 14 U14S, U14R and U14H chasses involves only a single 115 or 230VAC 50/60Hz input connection (see Figure 4-1A and 4-1B). (Figure 4-11 shows a detailed outline of this connection).

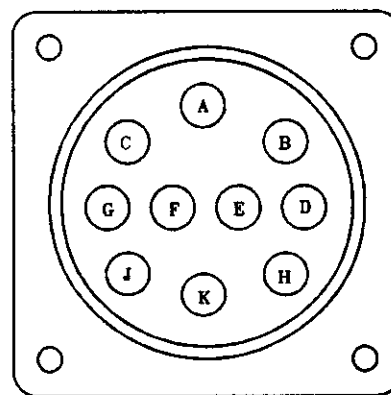
Motor output connections (up to four, depending on the type of chassis) consists of either Stepping Motor, Aerodrive or DC Servo Motor termination. Two types of motor output receptacles are available. An outline of these receptacles is shown in Figure 4-7.

Motor terminations to the receptacles shown in Figure 4-7 are of three types, two for Aerodrive and Stepping Motor operation, and one for DC Motor operation. Stepping Motor and Aerodrive termination can be of either "bipolar" wound or "unipolar" wound, as shown in Figures 4-8 and 4-9 respectively. DC Servo Motor termination is shown in Figure 4-10.

It should be noted that if the Unidex 14 is supplied with Aerotech motors and cables, you need not concern yourself with the detailed wiring termination shown in Figures 4-8, 4-9, 4-10 and 4-11.



14 Pin Plastic Style
(Standard)



10 Pin Metal Style
(Optional)

Figure 4-7: Motor Output Receptacles at Rear of U14S, U14R and U14H Chassis

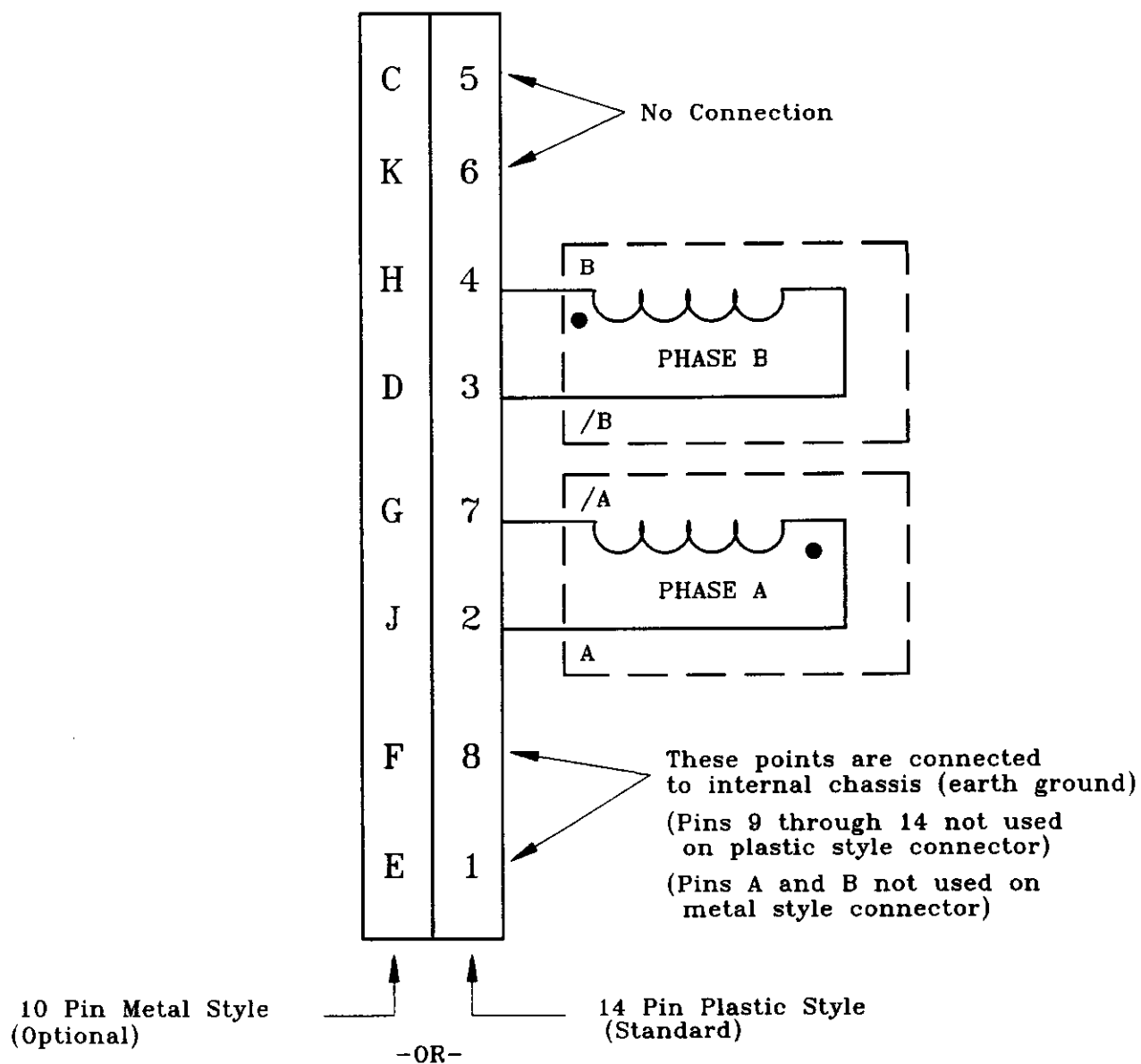


Figure 4-8: Outline of Connections for "Bipolar" Wound Stepping Motor and Aerodrive Motor

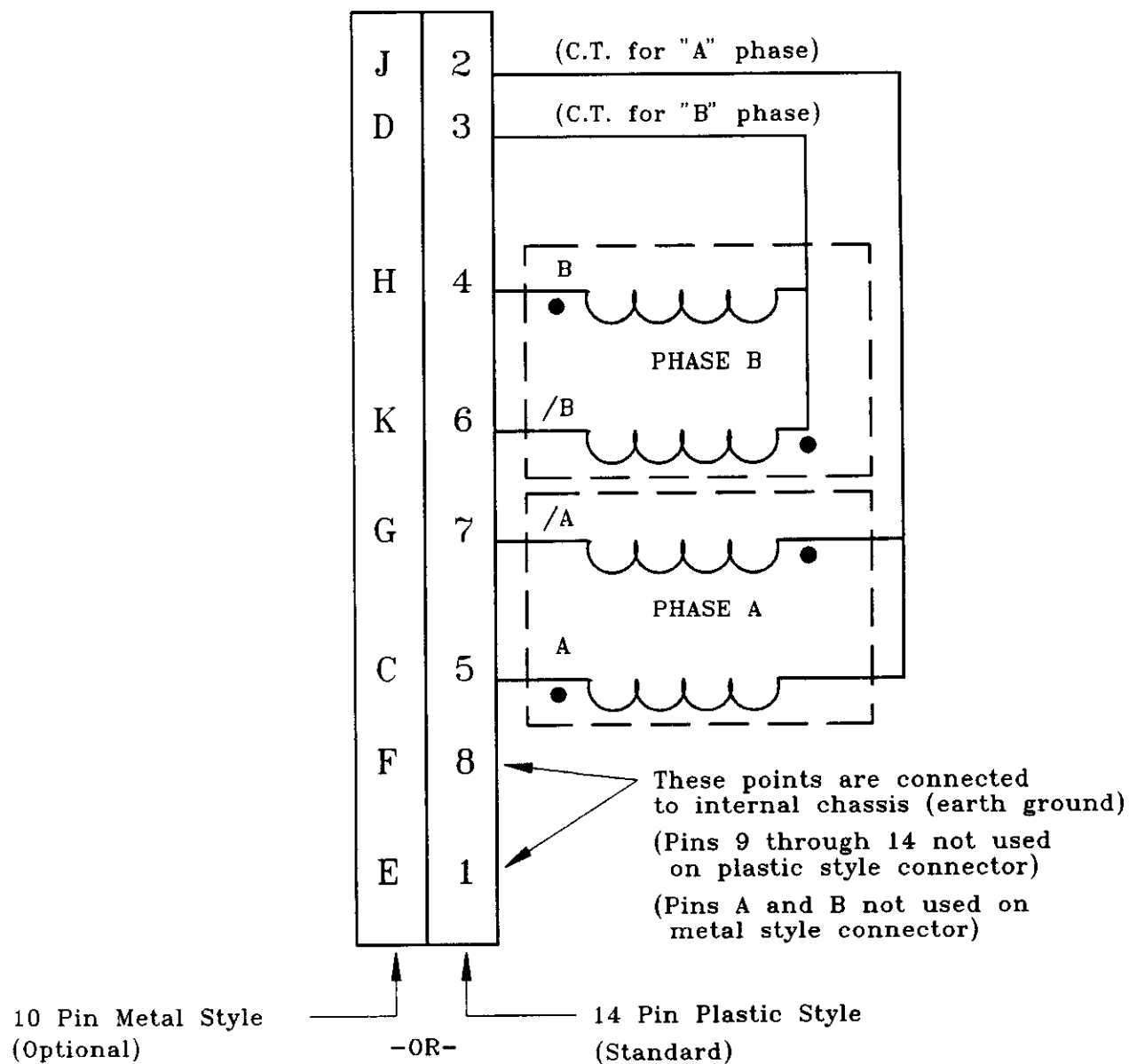


Figure 4-9: Outline of Connections for "Unipolar" Wound Stepping Motor and Aerodrive Motors

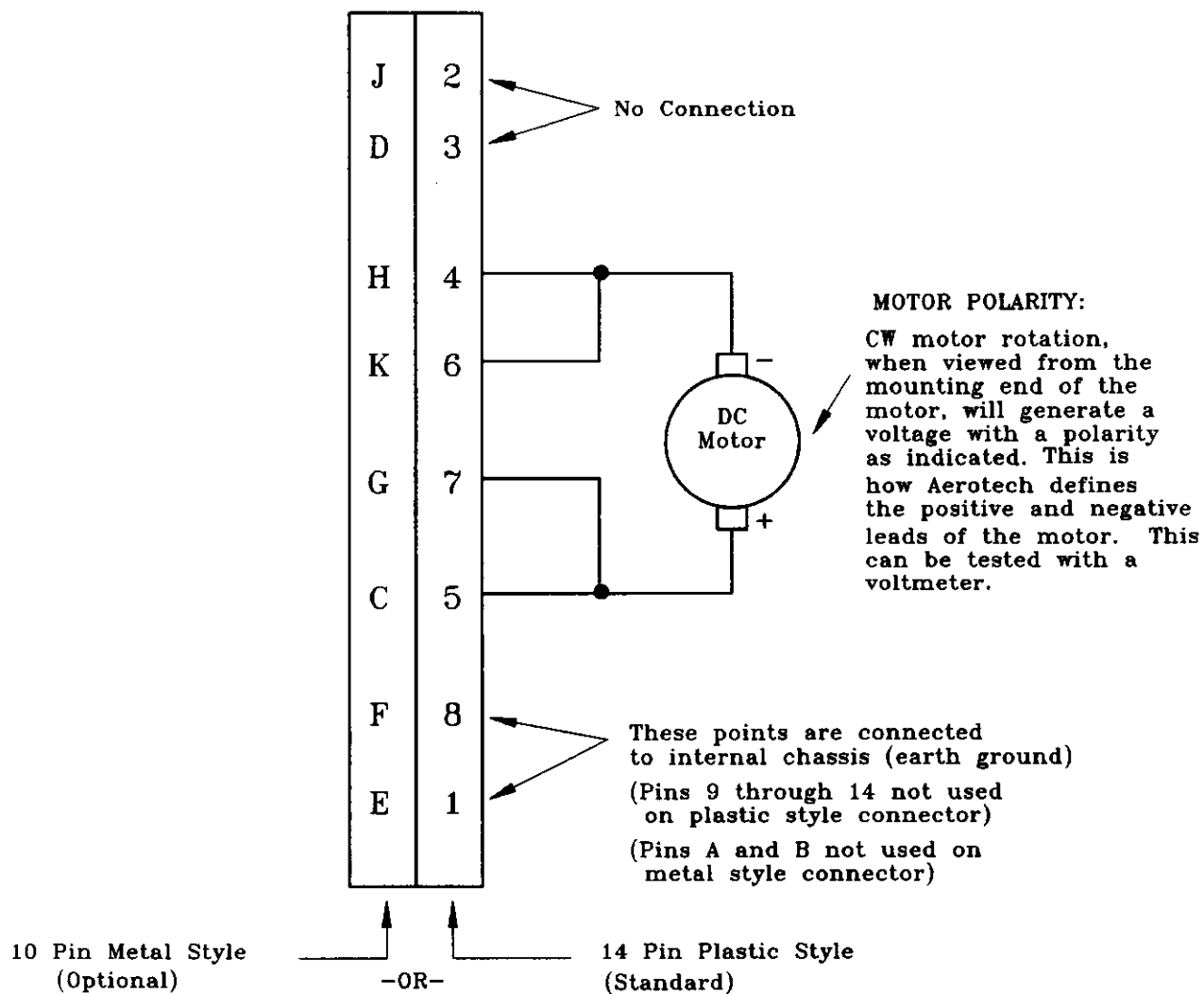


Figure 4-10: Outline of Connections for "Brush" Type DC Servo Motor

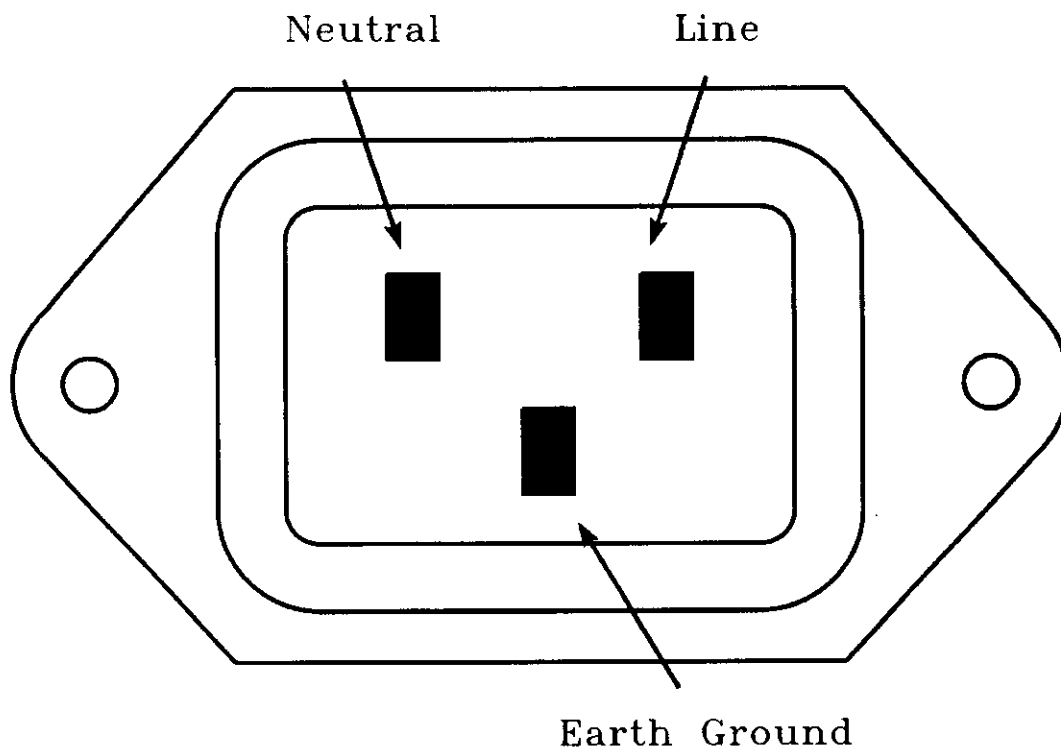


Figure 4-11: Outline of Input Power Receptacle at the Rear of Unidex 14 Chassis

CHAPTER 5: INTERNAL STRUCTURE OF UNIDEX 14

SECTION 5-1: DESCRIPTION OF U14S, U14R AND U14H DRIVE CHASSIS'S:

This chapter briefly discusses the internal hardware and drive module organization of the Unidex 14 U14S, U14R and U14H Chassis. This section contains general information regarding the placement into a given Chassis of Aerodrive, Stepping and/or DC Servo Drive modules (described in Chapter 4 of this Manual). Information on powerinterconnect wiring for the Drive modules in the chassis is also provided. For detailed information on specific Drive Modules (ie., DM4005, DAV16008, DSL8020 ect.), the U14 Control Interface Board, and their Power Supply Board, See Chapter 6 of this Manual.

A simplified wiring diagram of the U14S and U14R is shown in Figure 5-1. An accompanying outline of these Chassis with the front panel removed is shown in Figure 5-2. Figures 5-3 and 5-4 show similar drawings for the U14H chassis.

The U14S, U14R and U14H chassis provide a DC bus power supply for high power Stepping motor operation (using the DMV8008 and DMV16008) DC Servo motor operation (using the DSL3015, DSL4020, DSL8020 and DSL16020), and Aerodrive Motor operation (using the DAV4008 and DAV16008). The low power Stepping Drive modules (D and DM Series) can be used in these chassis without the need for a DC bus power supply (i.e., these modules internally generate control and bus power for the motor).

The DC bus power supply for the U14S, U14R and U14H chassis (see Figure 5-1 and 5-3) can be configured for a variety of DC voltage levels for use with the DMV8008, DMV16008, DAV4008, DAV16008, DSL3015, DSL4020, DSL8020 and DSL16020 Drive modules. The possible DC bus voltage levels for a given chassis is as follows:

U14S AND U14R CHASSIS:

DC BUS VOLTAGE	AXIS DISTRIBUTION
20VDC	All axes; X,Y,Z; Y,Z,T; X,Y; Z,T axes
40VDC	All axes; X,Y,Z; Y,Z,T; X,Y; Z,T axes
60VDC	All axes only
80VDC	All axes only

U14H CHASSIS:

DC BUS VOLTAGE	AXIS DISTRIBUTION
20VDC**	All axes, any single axis, or any axes group
40VDC**	All axes, any single axis, or any axes group
60VDC**	All axes, any single axis, or any axes group
80VDC**	All axes, any single axis, or any axes group
100VDC*	All axes only
120VDC*	All axes only
140VDC*	All axes only
160VDC*	All axes only

* These voltages are applicable to DMV16008, DSL16020 and DAV16008 operation only. 160VDC operation can be derived "off-line" (US/Domestic 115VAC line) without the use of transformer T1 or T2.

** Some combinations are only possible with 2 transformers (1500W option, 2P11-xxx).

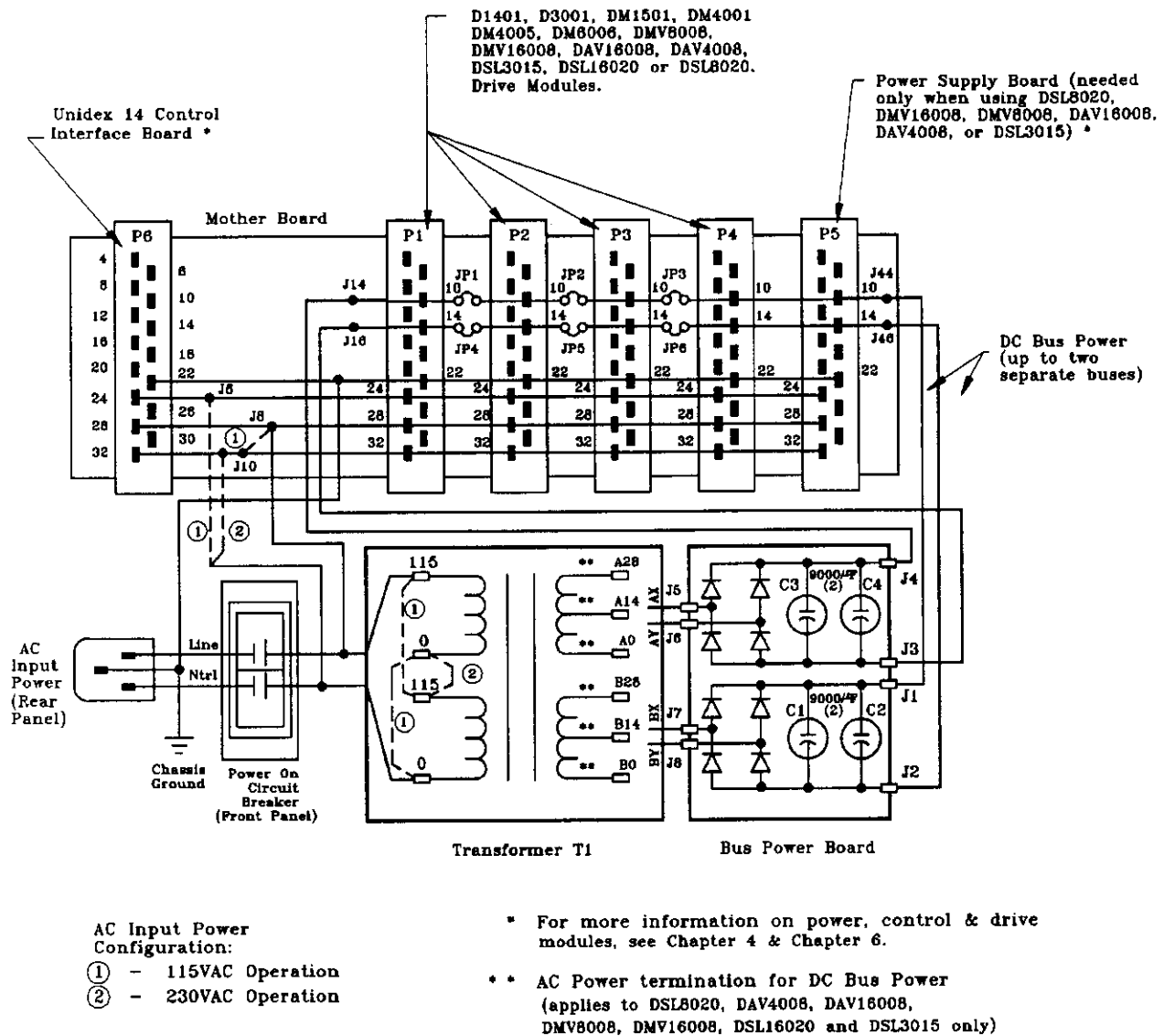
The secondary side of transformer T1 (and T2) provides the necessary AC voltage to be rectified to DC (see figures 5-1 and 5-3).

The individual transformer secondary windings A0-A14-A28, B0-B14-B28, etc., each supply up to 28VAC centertap voltages (A14, B14, etc. are the centertap connection).

Jumper connections on the motherboard (JP1, JP2, JP3, etc.) provide "connect/disconnect" points to the given axis for the selected DC bus voltages supplied by capacitors C1 through C4.

NOTE: DC Bus Power for the DSL3015 is handled differently than other drives. The DSL3015 as a "Class B" push-pull "Linear" amplifier requiring two DC Bus ("Plus" and "Minus"). The "Plus" bus is connected as shown in Figures 5-1 and 5-2. The "Minus" bus is connected to "Pin 4" of each drive slot (P1, P2, P3 and P4).

NOTE: The drive module slots (P1-P4, see Figures 5-1, 5-2, 5-3 and 5-4) are electrically interconnected so that damage will NOT occur to a given module or motor (if, for example, a stepping drive module is inadvertently plugged into a slot setup for DC servo operation). *Nevertheless, special care should be taken when removing the drive modules from the Unidex 14 chassis.*



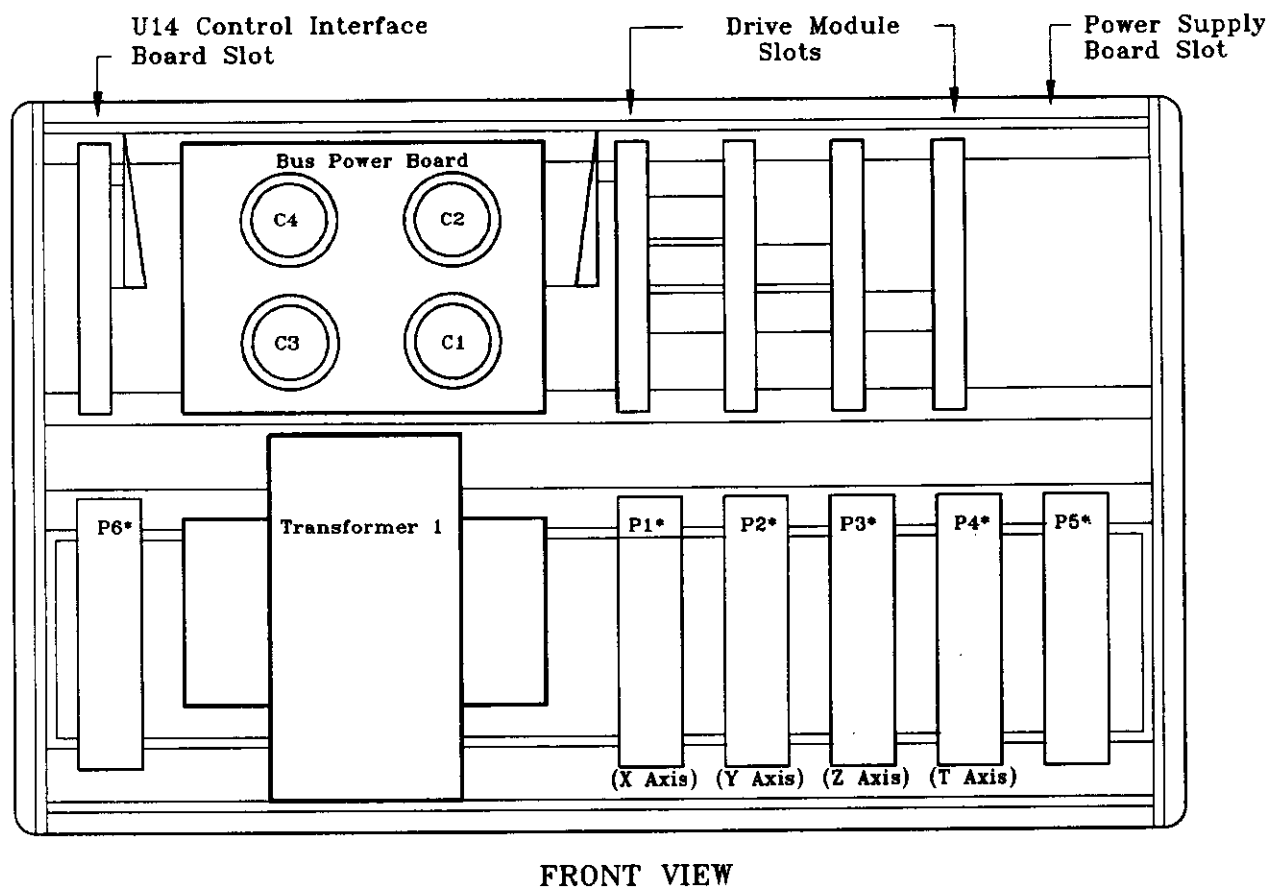


Figure 5-2: Outline of the U14S and U14R Chassis with Front Panel Removed

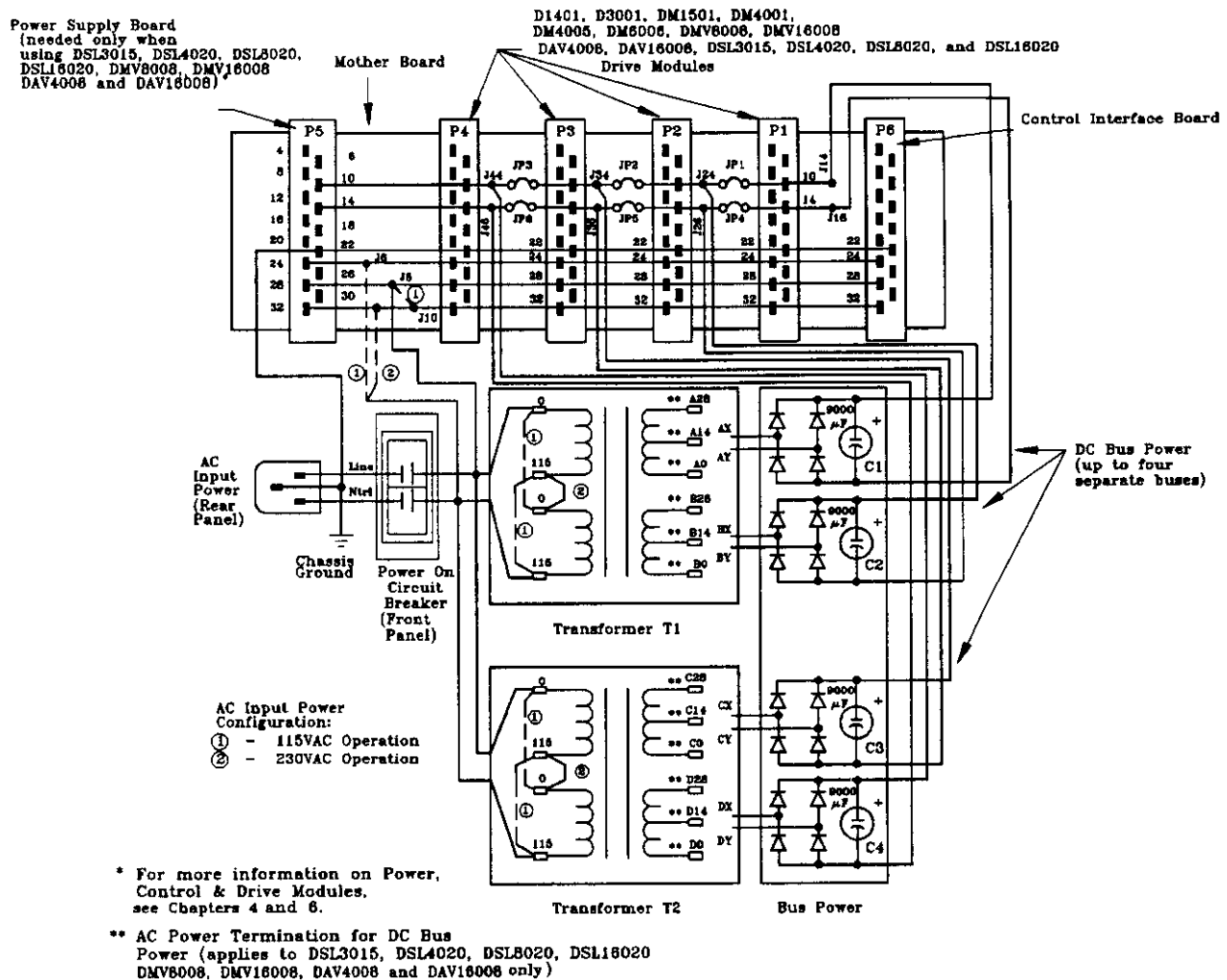


Figure 5-3: Simplified Wiring Diagram of U14H Chassis with Top Cover Removed (see Figure 5-4 for Outline)

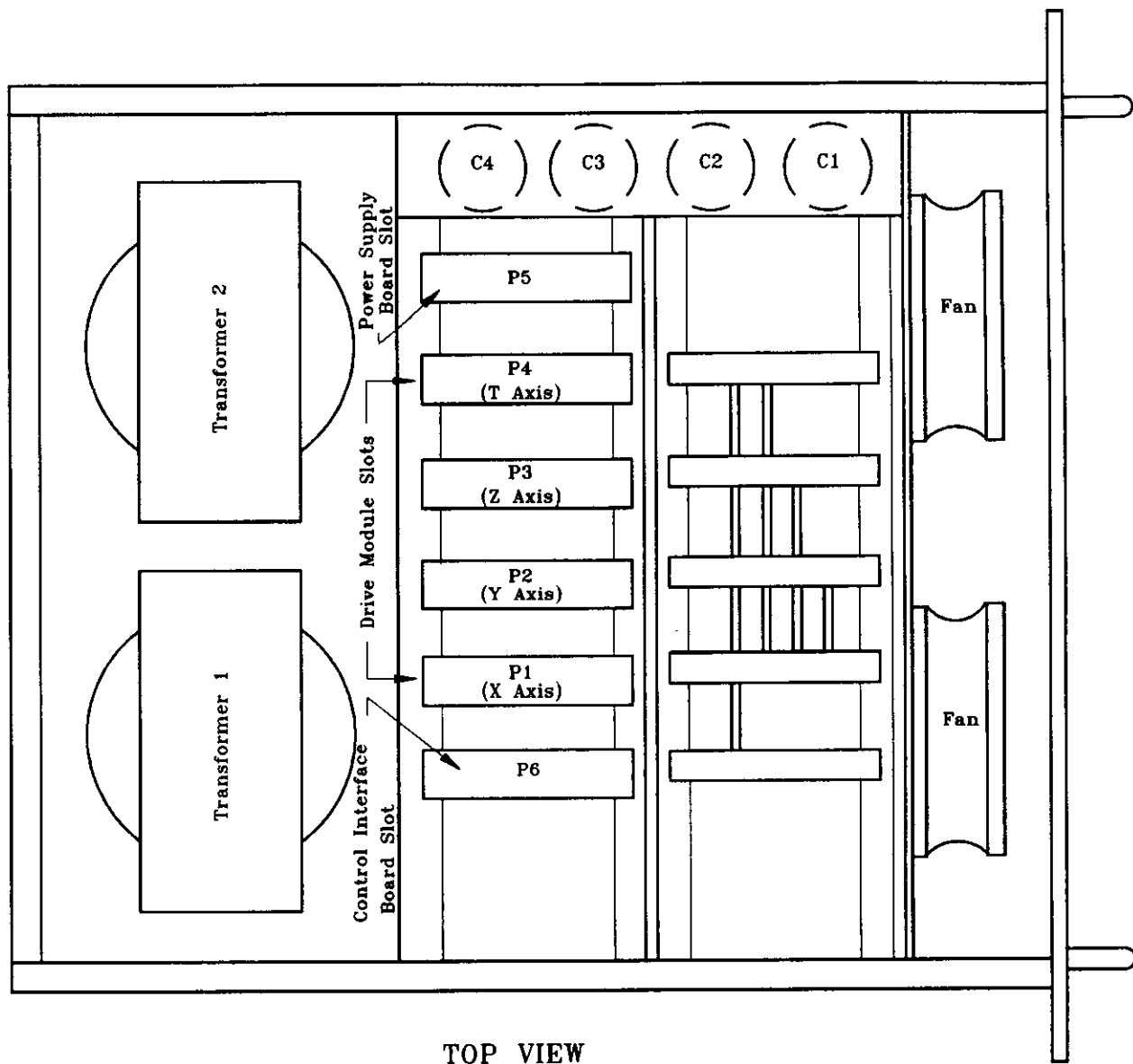


Figure 5-4: Outline of the U14H Chassis with Top Cover Removed

SECTION 5-2: DESCRIPTION OF THE U14C CONTROL BOARD

A schematic of the U14C Control Board is shown in Figures 5-5 thru 5-23. A simplified block diagram of this board is shown in Figure 1-1, Jumper locations are shown in Figure 2-1.

Normally the user need not be concerned with the details of these schematics. However, the general description of the schematics which follow, may aid in understanding the Unidex 14 Control System.

The 68000 microprocessor on the U14C Control Board maintains four concurrent processes. The highest priority process calculates the desired pulse frequency 1024 times each second with a proprietary algorithm (patent number 4,734,847). This number is fed to U62, U63, U64 and U65 which synchronizes the motor pulses to the microprocessor and actually generate the pulse train. The velocity profile and synchronization of each axis is also handled by the 68000.

The commands from the PC/XT/AT or compatible host computer are temporarily stored in a 124 character buffer until the 68000 microprocessor can parse them. The command is then executed immediately or routed to separate command queues for each axis. The command queue contains a list of addresses to execute followed by an optional parameter. A command from the host may be expanded into several commands to the appropriate axis. The GO command for example will expand into start, ramp up, constant velocity and ramp down commands. The LS command will save its parameter, i.e. the loop count, on a loop stack along with the address of the LS command to be used by the next LE command as a target for a jump command. The LE command will decrement the loop count and jump to the most recent LS command providing the loop count has not reached zero. If the loop count has reached zero and it is not nested inside another loop, the queue space will be flagged as available and the next instruction in the queue will be executed.

Interrupts to the PC/XT/AT host are latched by U33. The enable status of each interrupt is also stored by U33. Status of the interrupts and error flags may be read by the host via U33. U12 latches the interrupt request enable bits and allows them to be read back by the microprocessor. U16 compares the address to the I/O address selected by the host and enables the board decode logic when a match is detected.

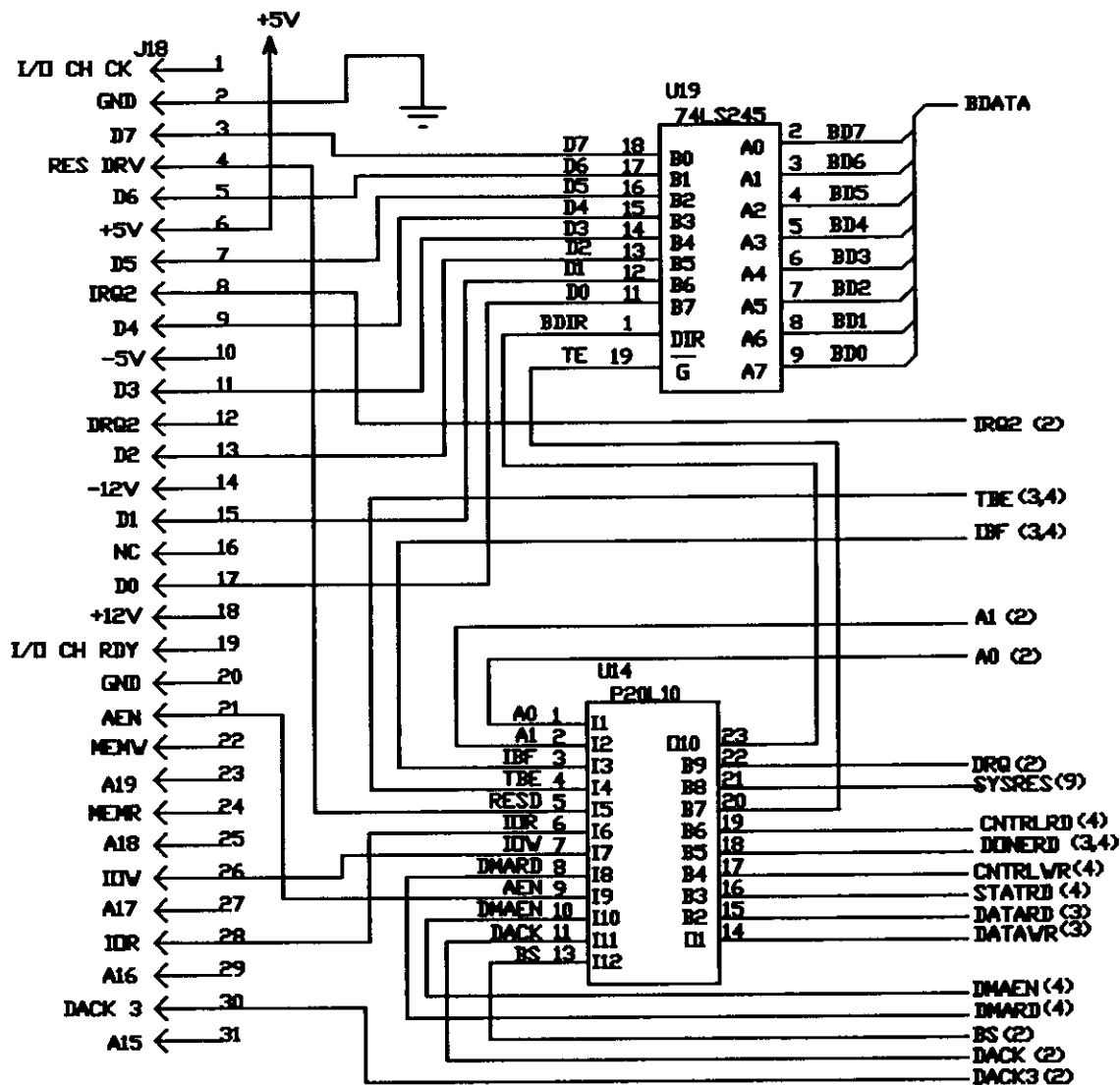


Figure 5-5: U14C Control Board Schematic

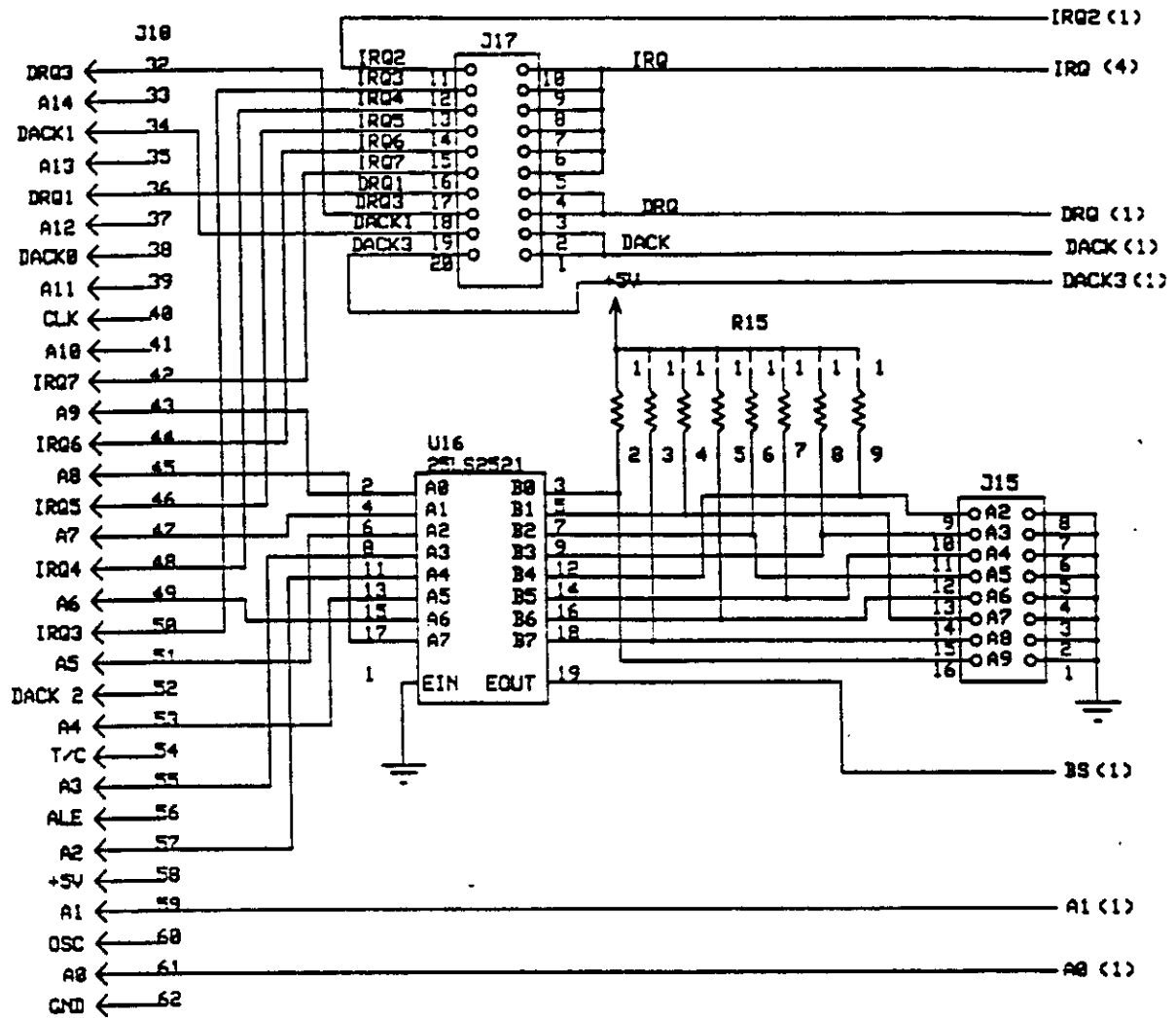


Figure 5-6: U14C Control Board Schematic (con't)

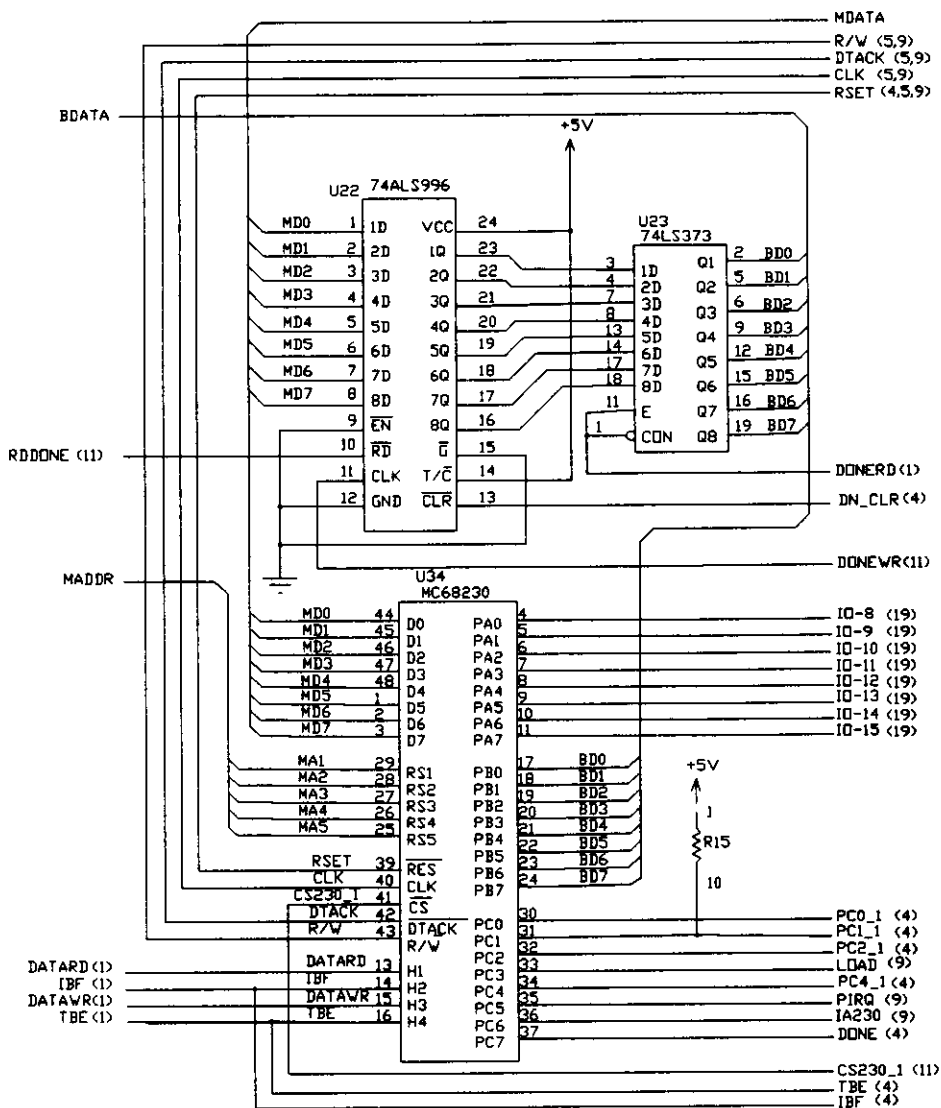


Figure 5-7: U14C Control Board Schematic (con't)

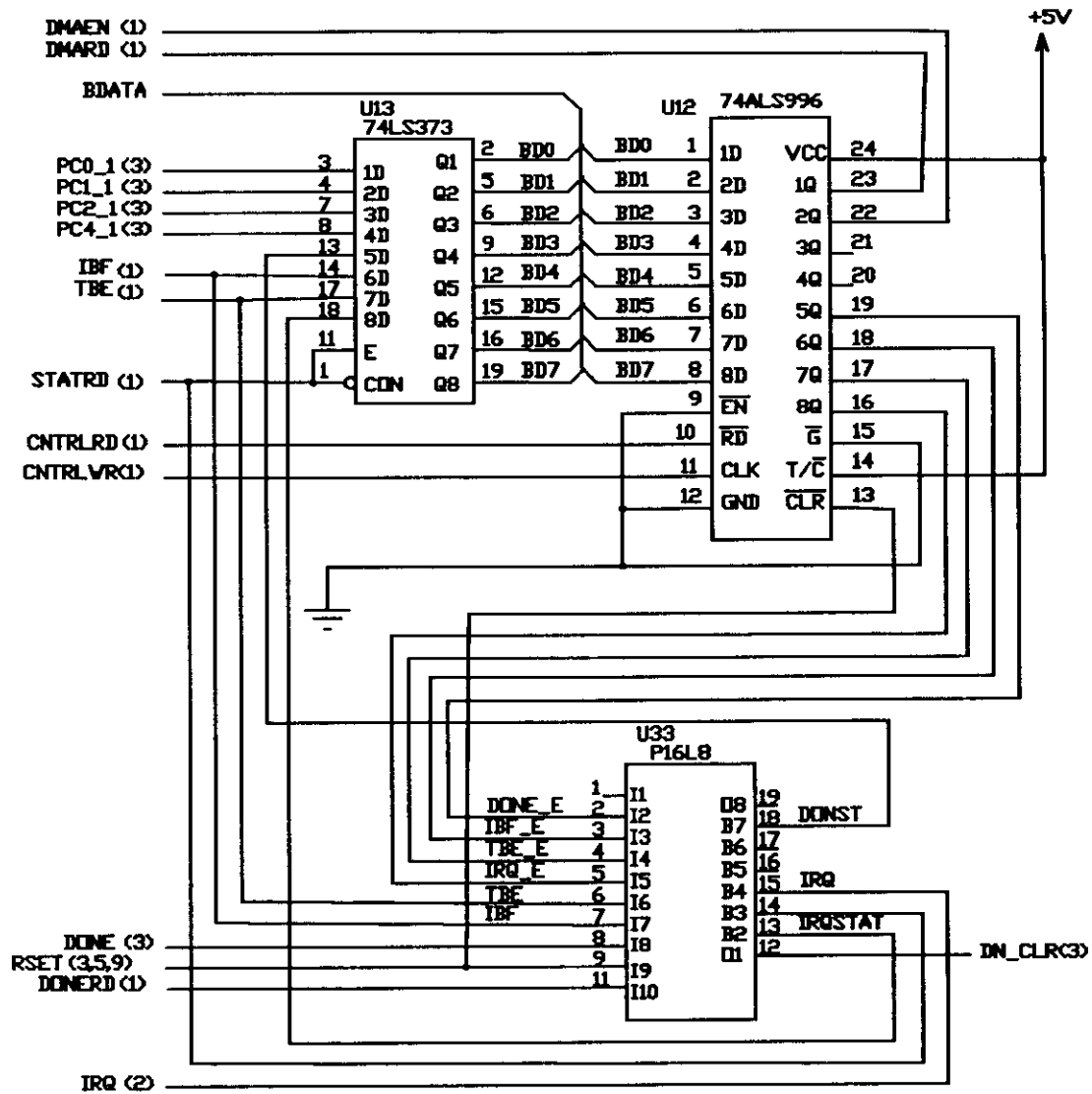


Figure 5-8: U14C Control Board Schematic (con't)

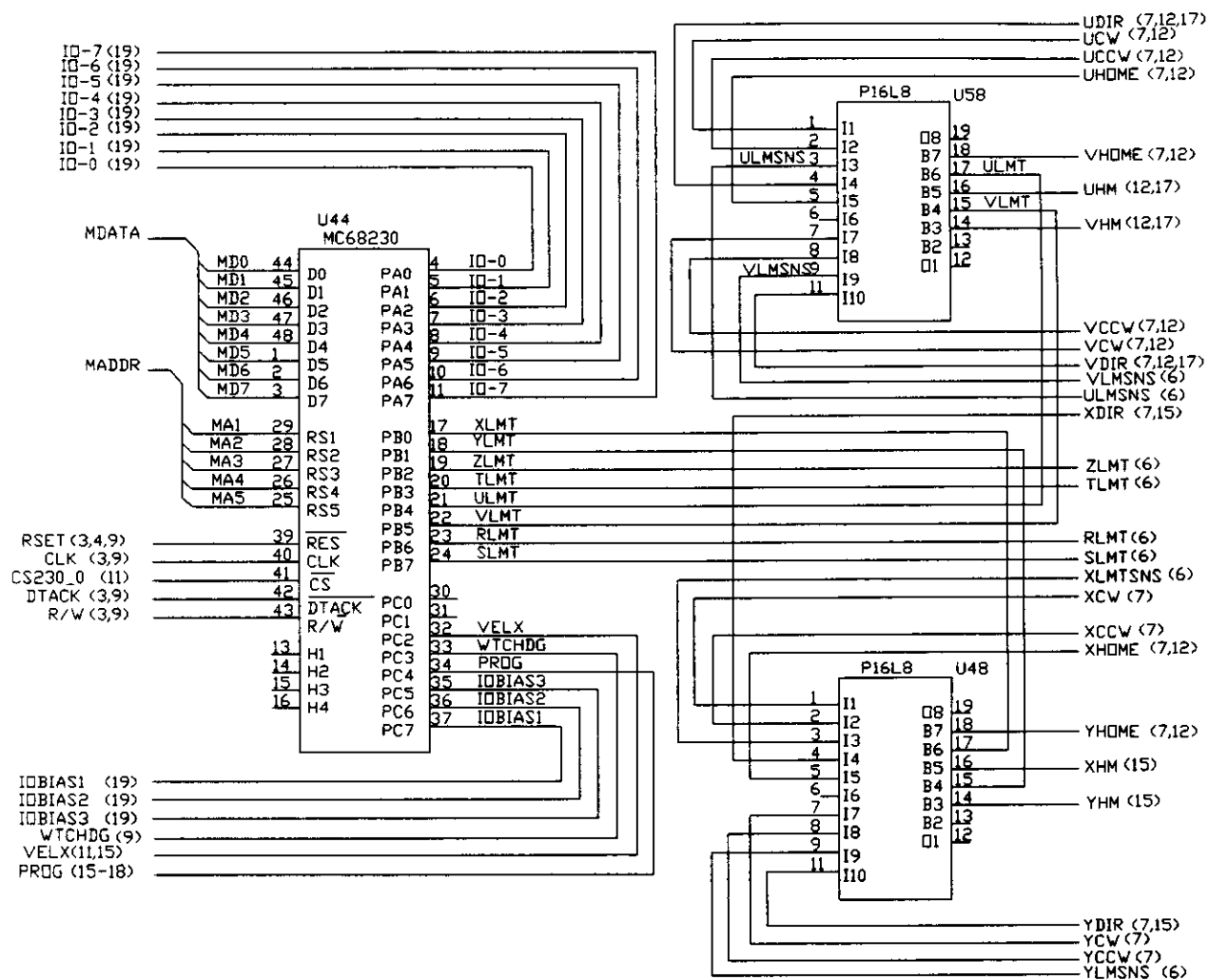


Figure 5-9: U14C Control Board Schematic (con't)

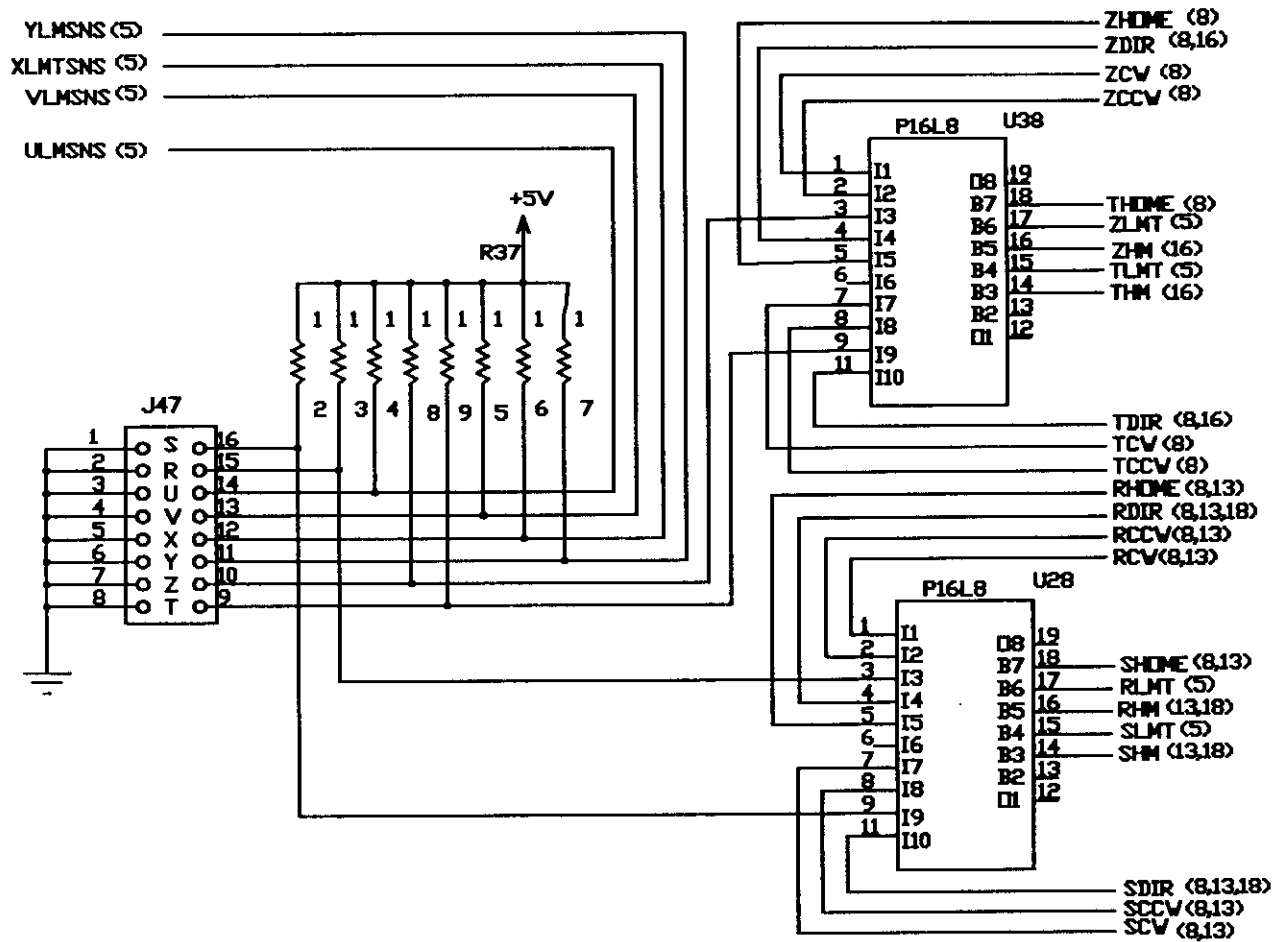


Figure 5-10: U14C Control Board Schematic (con't)

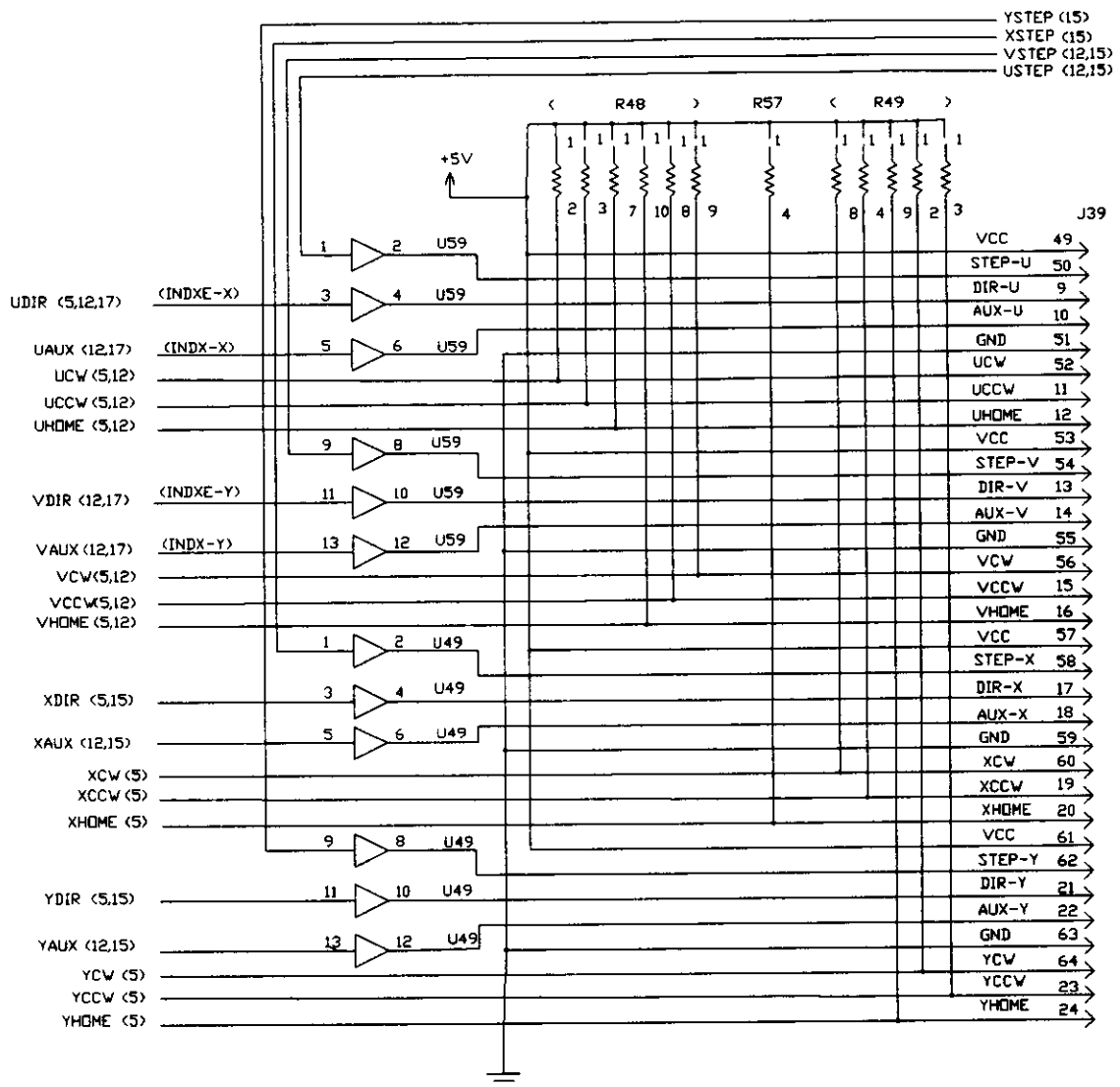


Figure 5-11: U14C Control Board Schematic (con't)

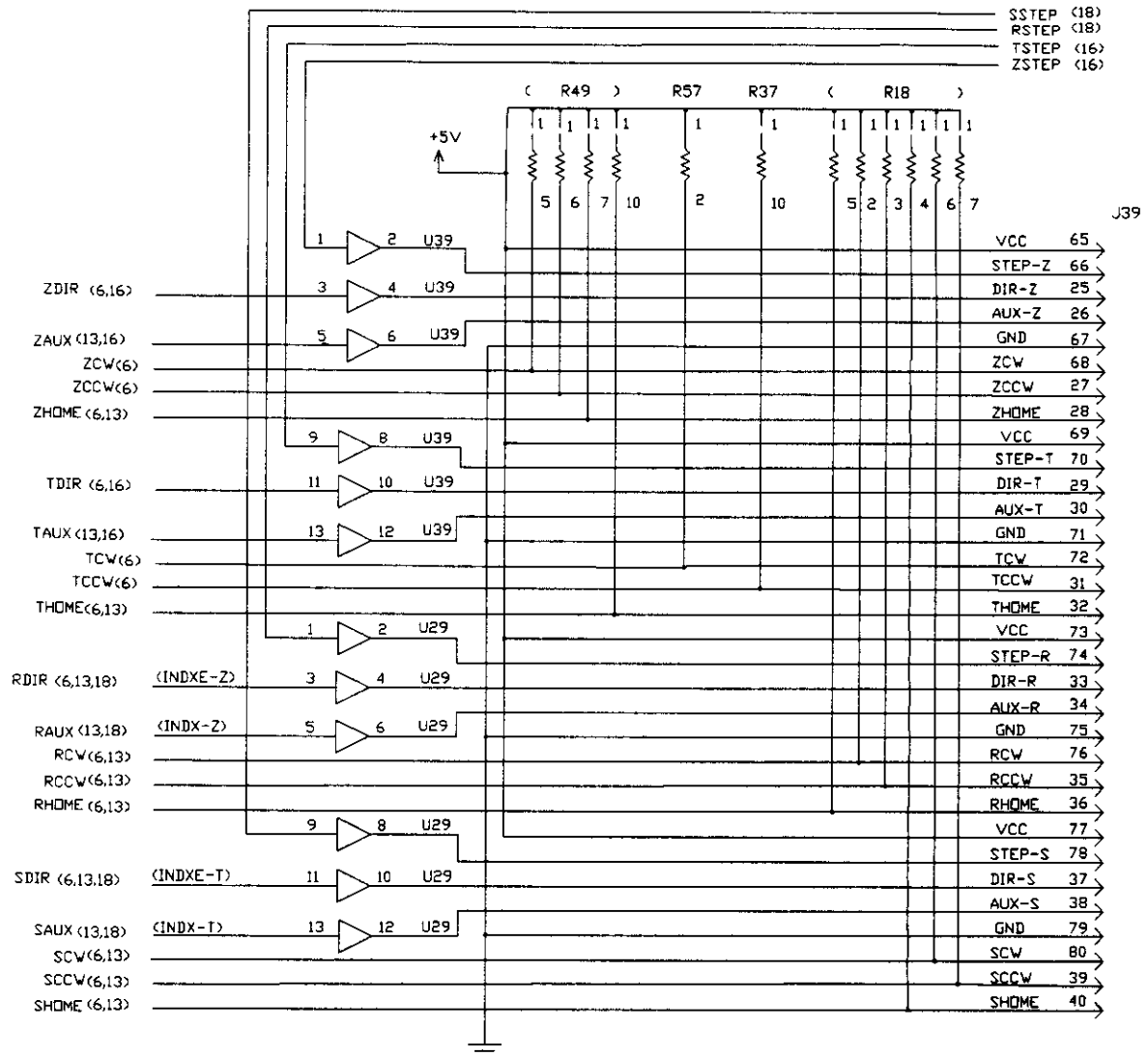


Figure 5-12: U14C Control Board Schematic (con't)



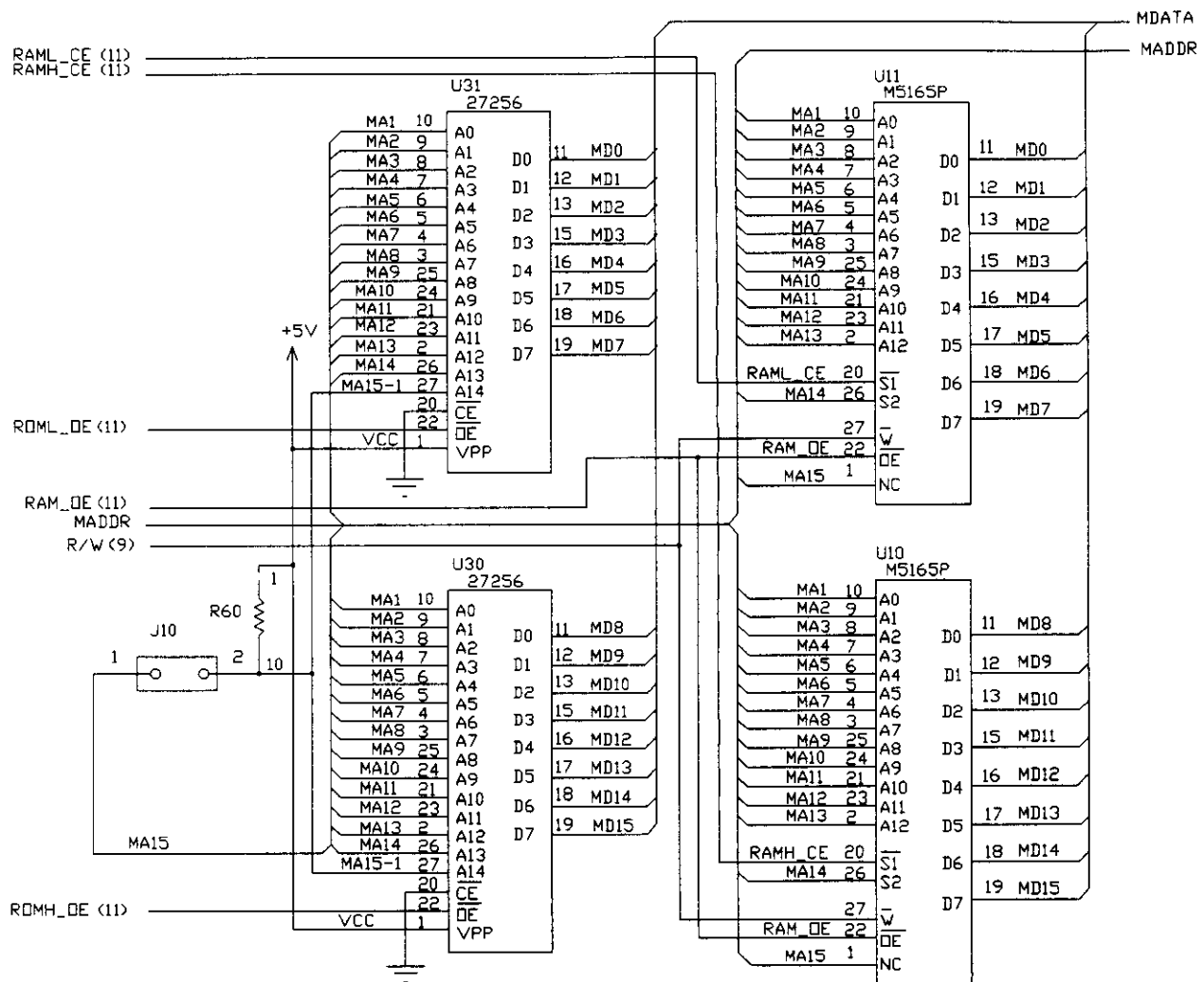


Figure 5-14: U14C Control Board Schematic (con't)

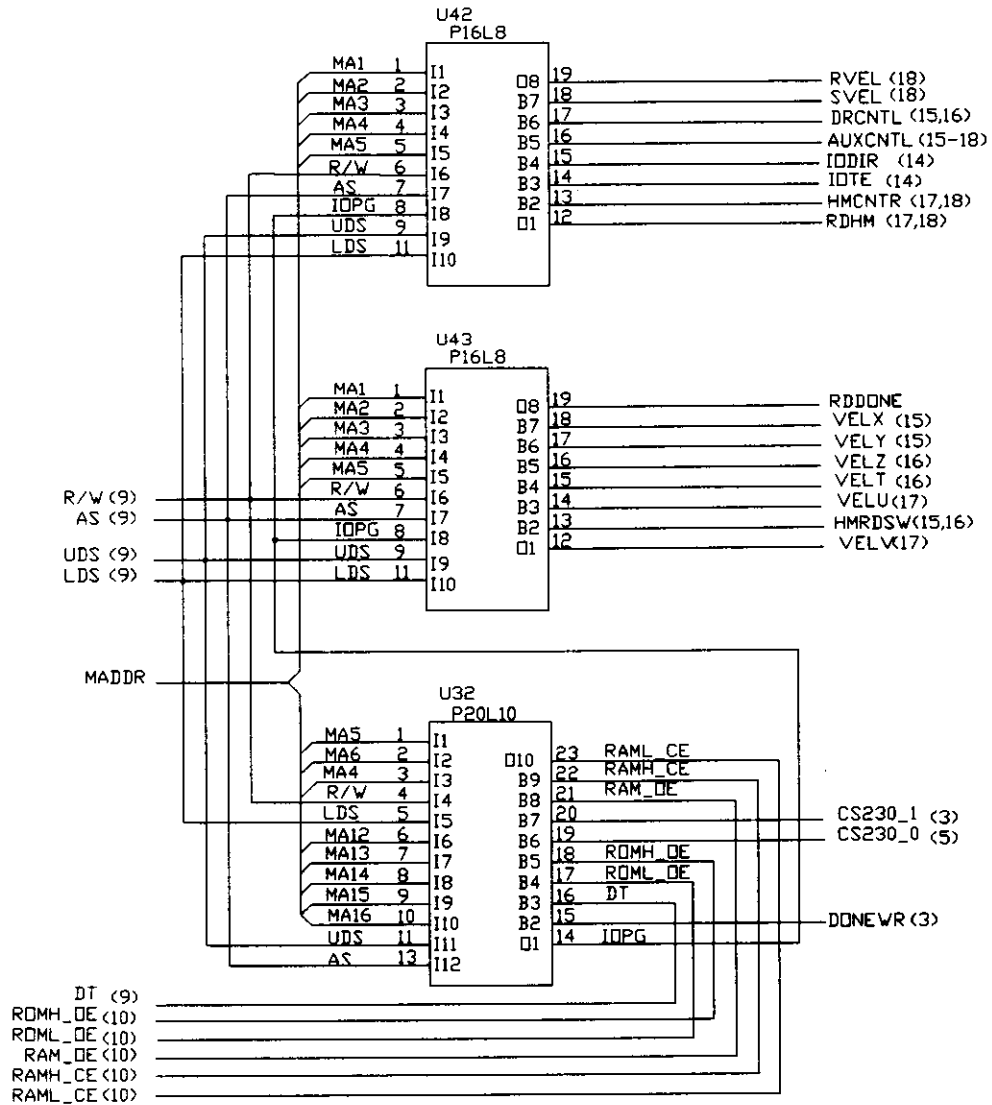


Figure 5-15: U14C Control Board Schematic (con't)

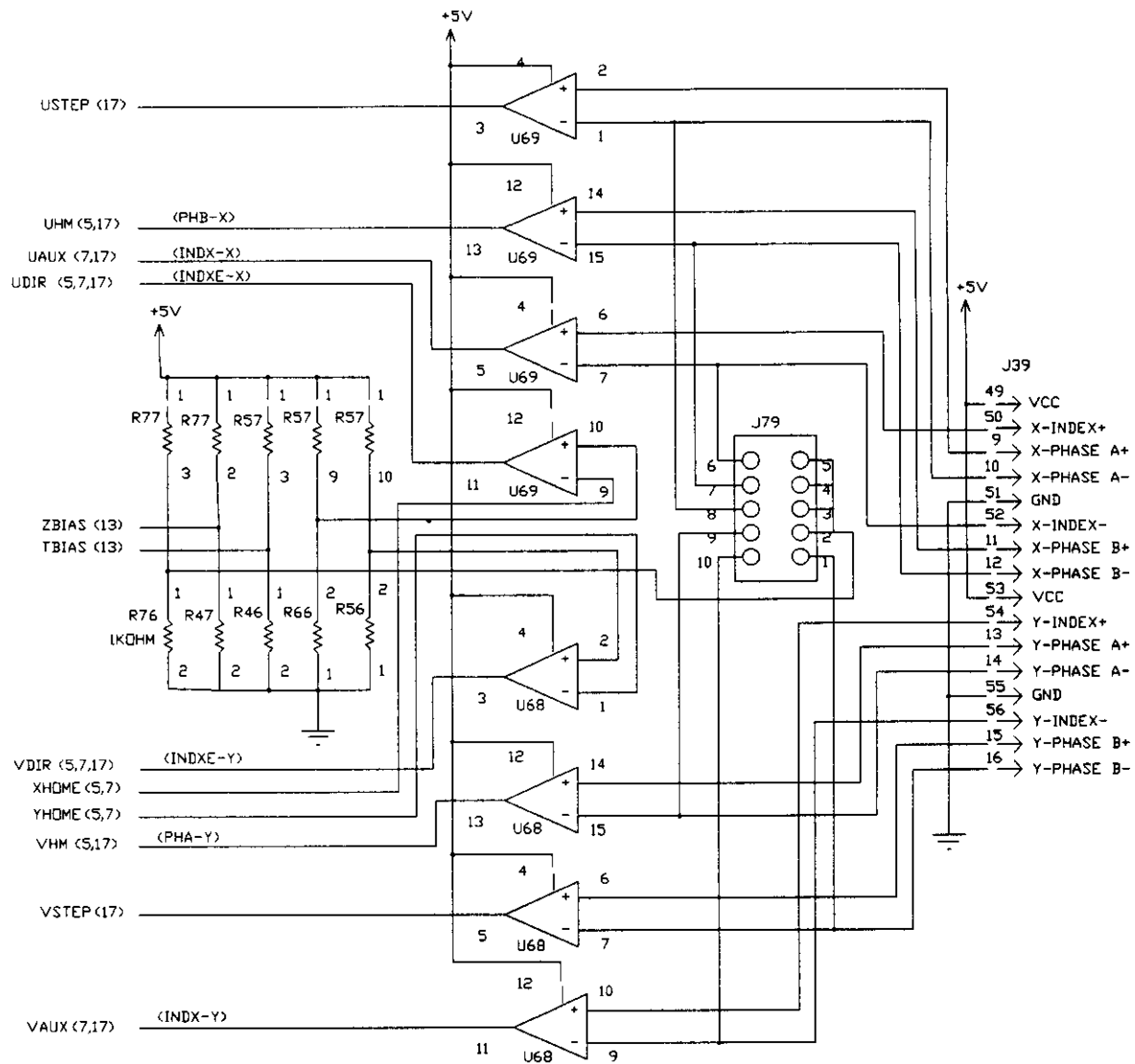


Figure 5-16: U14C Control Board Schematic (con't)

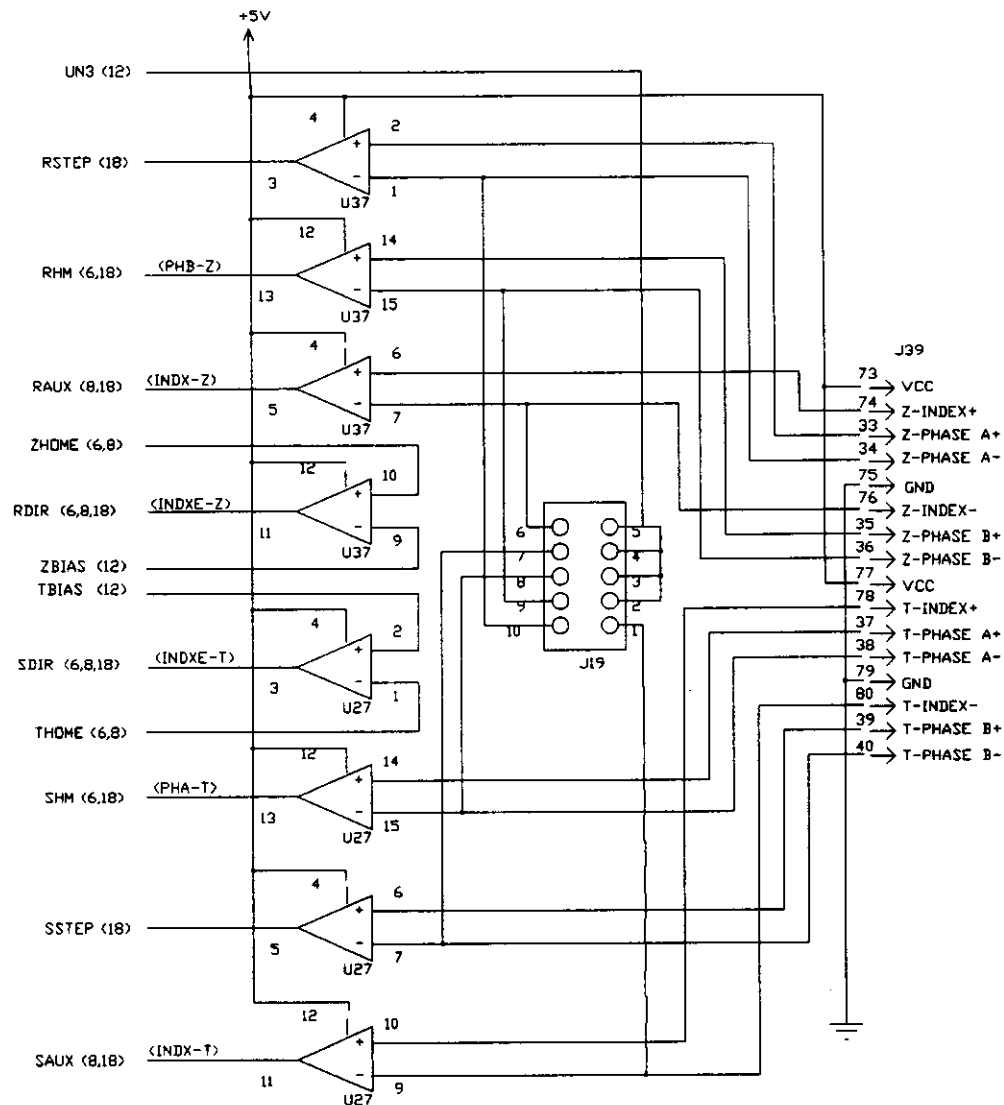


Figure 5-17: U14C Control Board Schematic (con't)

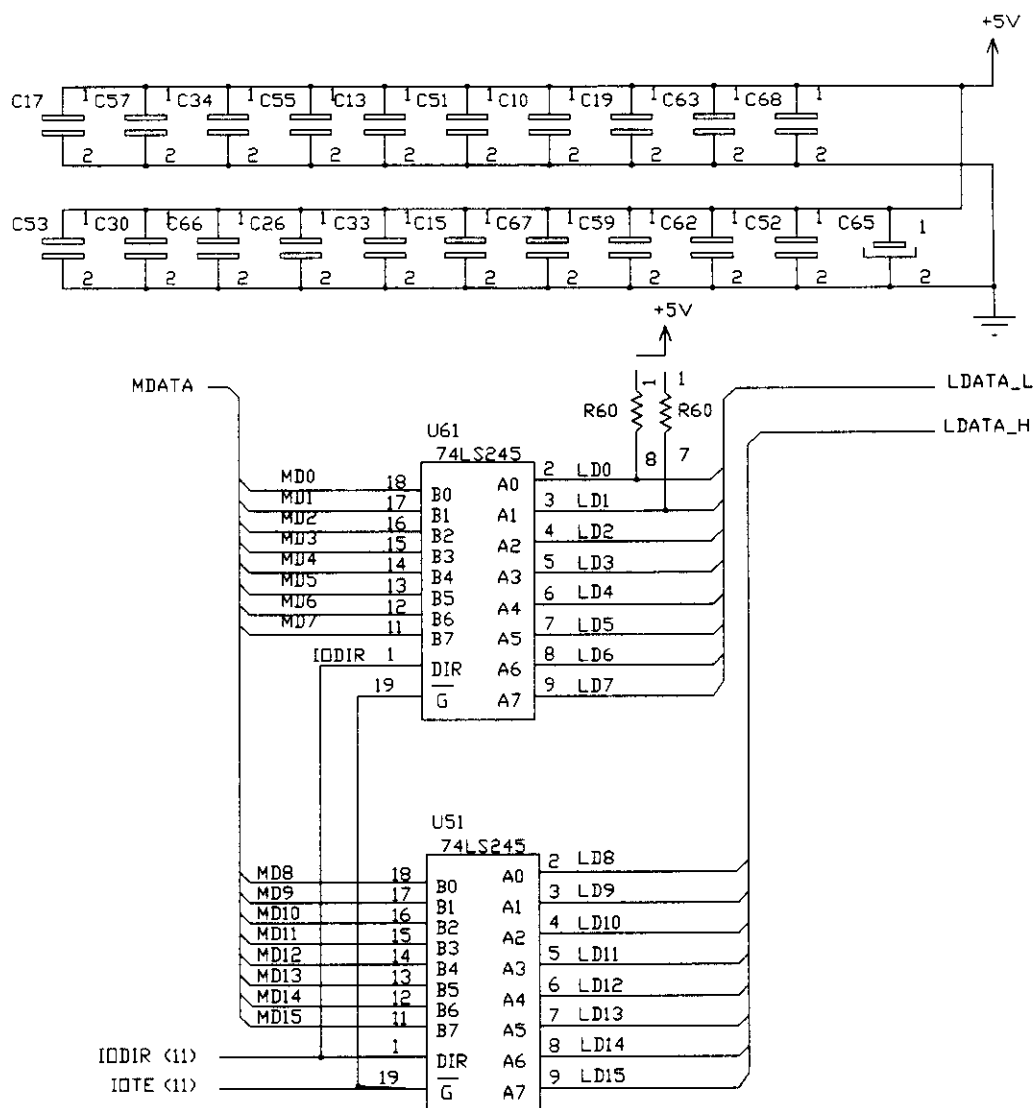


Figure 5-18: U14C Control Board Schematic (con't)

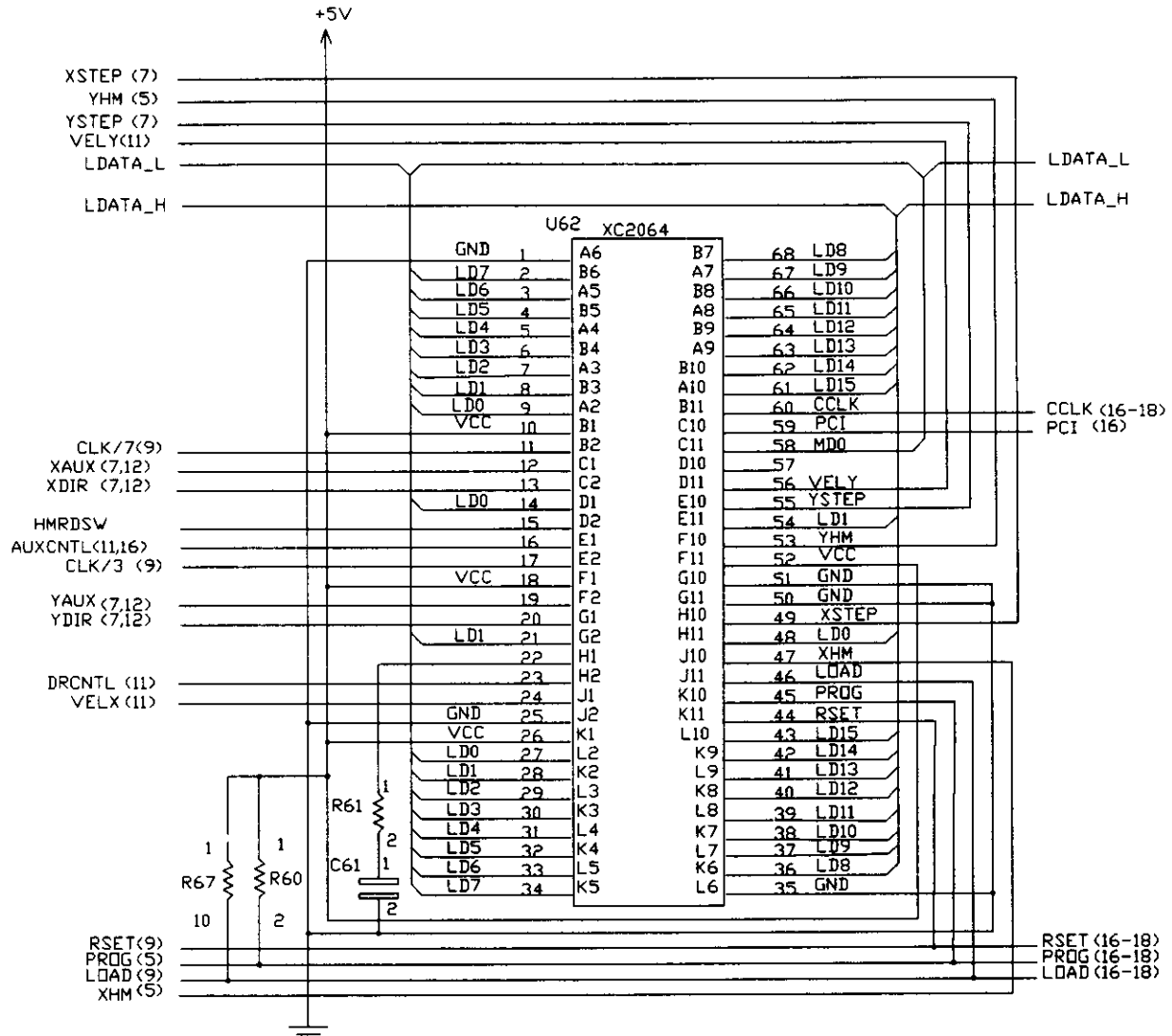


Figure 5-19: U14C Control Board Schematic (con't)

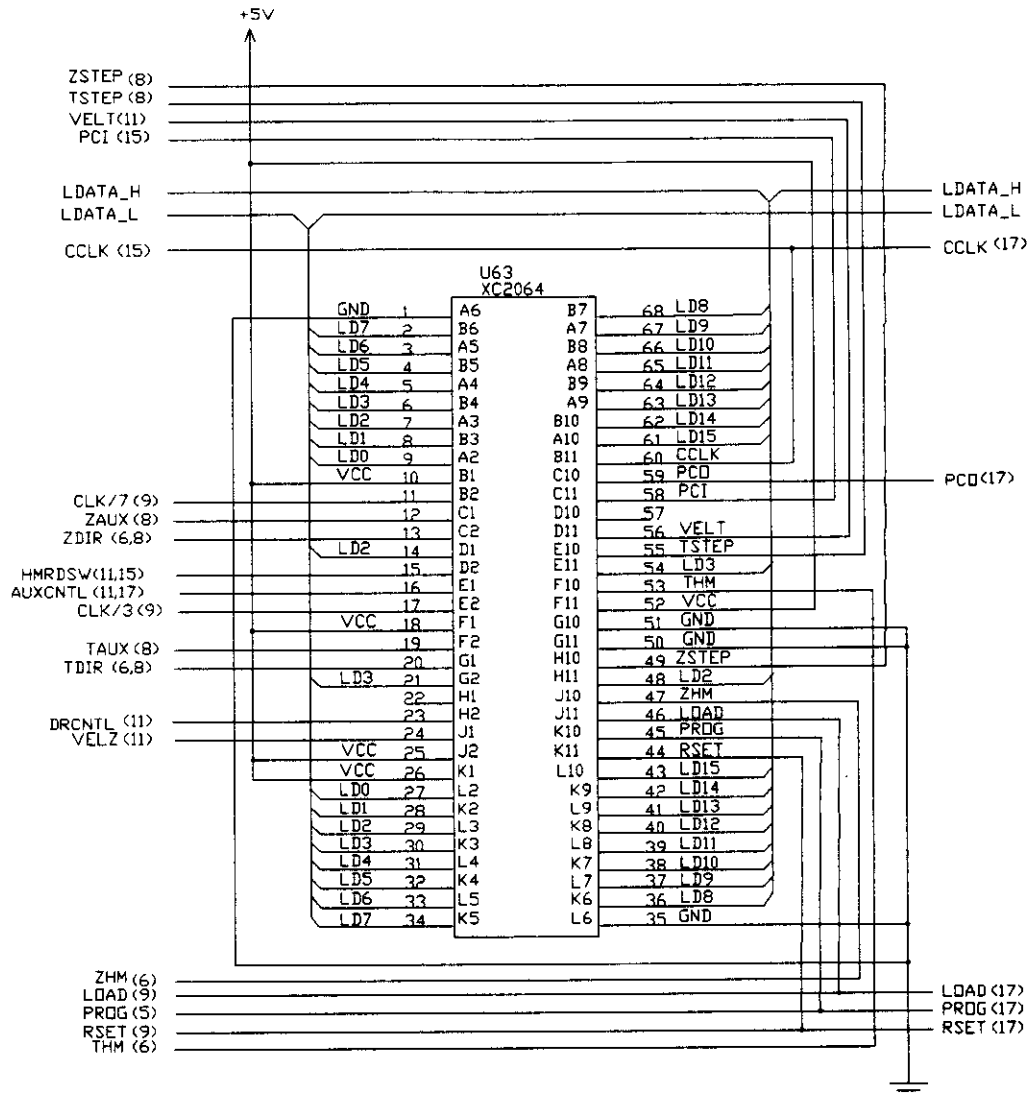


Figure 5-20: U14C Control Board Schematic (con't)

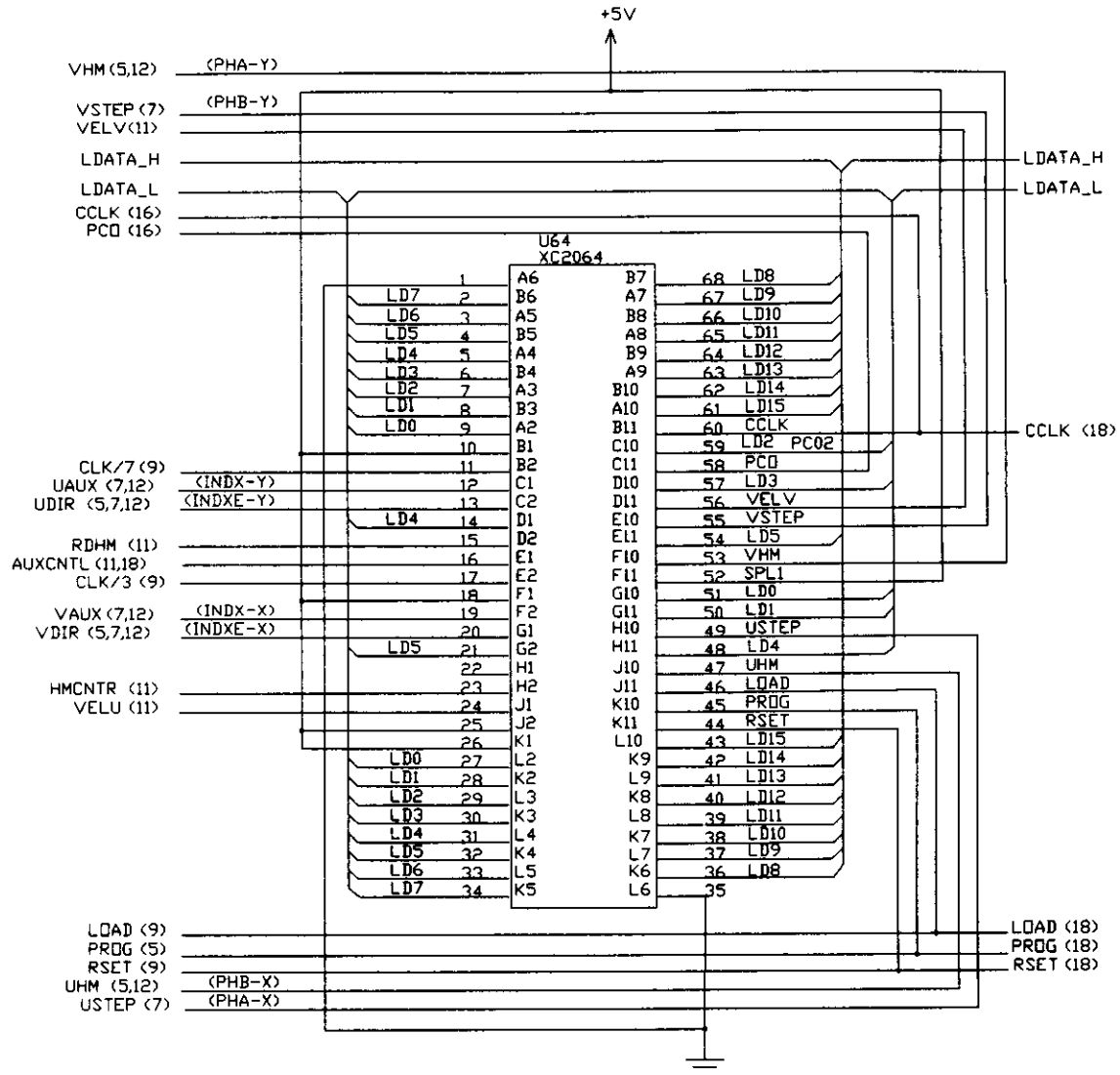


Figure 5-21: U14C Control Board Schematic (con't)

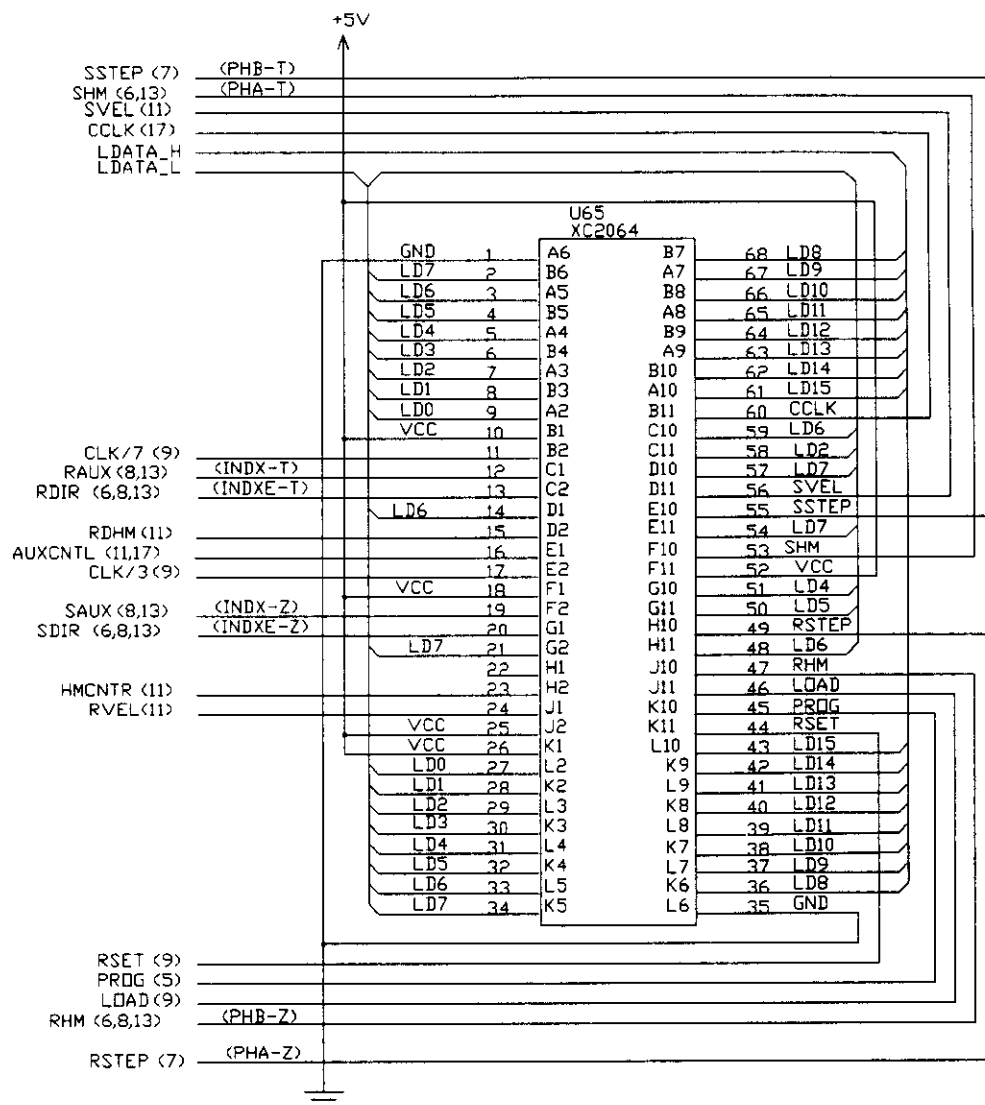


Figure 5-22: U14C Control Board Schematic (con't)

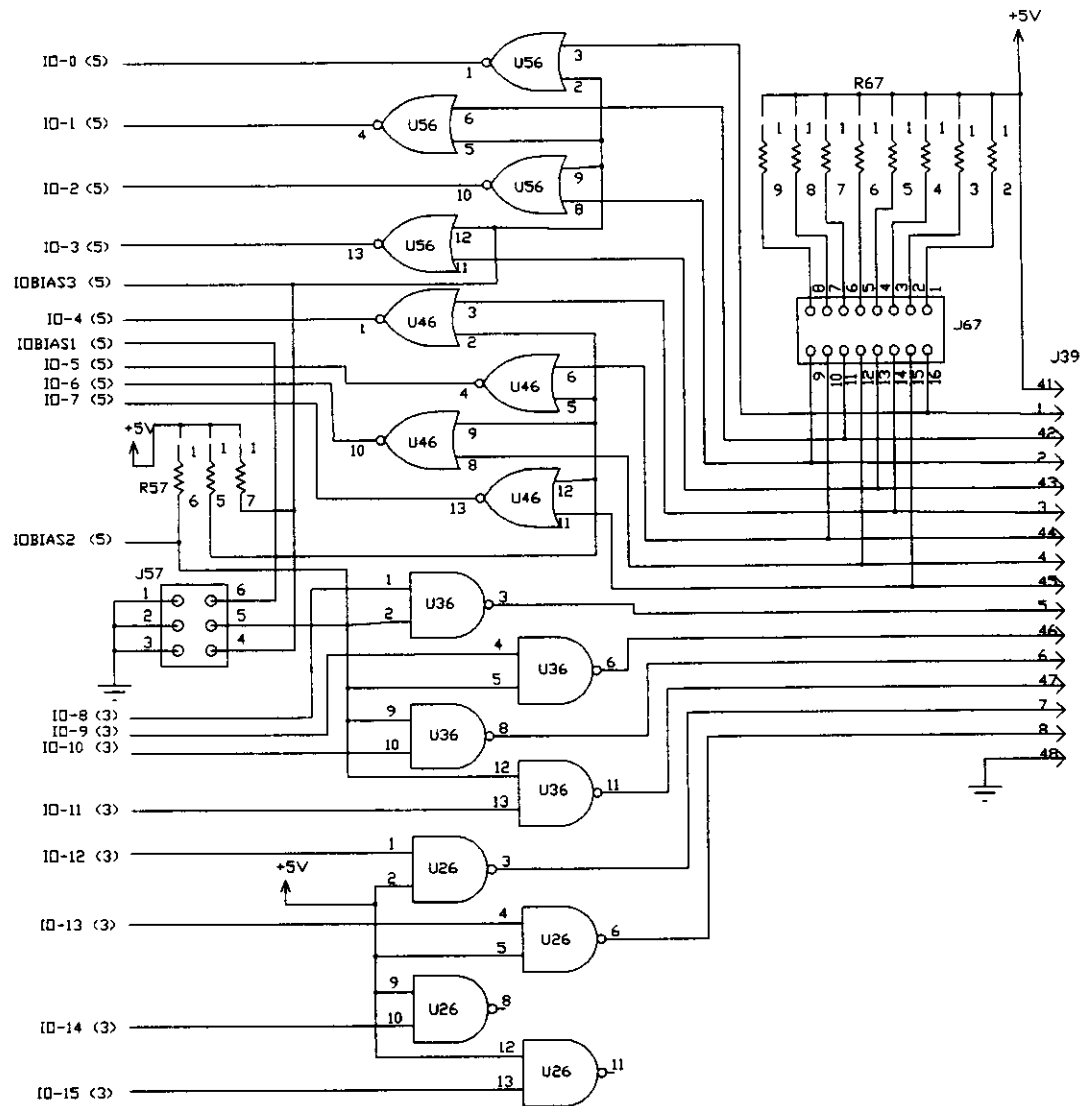


Figure 5-23: U14C Control Board Schematic (con't)

CHAPTER 6: DRIVE MODULES

CONTROL AND POWER BOARD SPECIFICATIONS

Specifications for the Stepping Motor drive modules, DC Servo Drive modules, Aerodrive modules, Power Supply module, and Control module are described in this section.

Information provided for each module described in the following section is limited to the description of the various operational alterations that may be performed on a given module. An example of an alteration may be the re-selection of the "Homing" direction from CCW motor rotation to CW motor rotation.

IMPORTANT: Throughout this manual whenever reference is made to CW and/or CCW motor rotation, it is assumed that the user is "looking into" the motor mounting flange.

SECTION 6-1: STEPPING DRIVES

6-1-1: DM4001 and DM4005 STEPPING DRIVES

6-1-1-1: CIRCUIT DESCRIPTION

The DM4005 and DM4001 are designed to accurately control a standard unipolar Stepping motor by utilizing a unique sin/cos Translator control. This control is capable of dividing a full step size into 250 evenly spaced micro steps. For a 50 pole 200 full-step Stepping motor, this subdivision of a full step provides electrical resolutions of up to 50,000 steps per revolution. This sin/cos translator control, can be easily altered to provide other step sizes. By simply programming two IC chips, a host of other incremental step sizes can be chosen, ranging from 200 steps/rev (full step) up to 50,000 steps/rev in 200 step intervals. In other words, there are 250 choices of stepping resolutions.

Another feature unique to the DM4005 and DM4001 is the totally integral power supply circuit. Stepping motor voltages as well as all logic control voltages (+5 and $\pm 12\text{VDC}$) are generated on the board. The only incoming power connection necessary is a standard 115VAC 50/60 Hz (nominal) or a 230VAC 50/60 Hz line connection. Thus, the DC bus power supply subsystem of the U14S and U14R chassis (transformer T1 and/or T2 with capacitors C1, C2, C3 and C4, see Figures 5-1 and 5-2) is not required. The power supply board module (described in this chapter) is *not* required when using this drive module.

An outline of the DM4001 and DM4005 drive modules is shown in Figure 6-1. Table 6-1 provides specifications for the DM4001 and DM4005. Torque/Speed curves are provided in Figure 6-2.

This module can be used in an Unidex 14 chassis (U14S and U14R).

6-1-1-2: "HOME" REFERENCE DEFINITION AND OPTIONS

All Unidex 14 Controllers are capable of generating a cold start reference position, which is the "Home" position. The basic Home cycle involves the following series of events. When the "Go Home" command has been issued, the motor will turn CCW (standard) or CW (optional) until a "Home Limit Switch" activation occurs.

NOTE: The CCW and CW rotational references are viewed, "looking into" the motor mounting flange.

Upon "Home Limit Switch" activation, the motor reverses and rotates in the opposite direction until the switch deactivates. If a "Marker" pulse option does not exist, upon switch deactivation, the motor will reverse and stop. If a "Marker" pulse option does exist (see Section 6-1-1-3), the motor reverses and continues to rotate until the "Marker" pulse is located and then stops.

The speed at which the Home cycle occurs is factory set at 120 RPM. If a different speed is required, see Section 6-1-1-7 for appropriate Personality module changes.

For most rotary motion stages, the "Home Limit Switch" referenced above is an independent switch incorporated specifically for the "Home" cycle. For linear motion stages, the "Home Limit Switch" could be an independent switch as well. However, in most cases, the CCW or CW limit switches perform "double duty" and act as the "Home Limit Switch". (Note that the process of putting the "Home Limit Switch" input in parallel with the CCW or CW limit switch input is standardly done at the motor/stage, external to the Unidex 14.)

The "Home Limit Switch" wiring to either a CW or CCW limit switch is done at the motor/stage external to the Unidex 14.

If it is not possible to use the "Home Limit" input at J13, 14, 19, or 20, such as when Opto 22 coupled limits are being used, the "CCW" or "CW limit" input may be paralleled to the "Home Limit" input by reconfiguring jumpers 16 through 19 on the DM4005/DM4001 PC board. Note that this scheme is possible only with linear stages and not with rotary states.

The jumper definition follows:

HOME LIMIT SOURCE:

(Standard) From Home Limit Input: Jumper 17-19, Remove 16-19,18-19

(Optional) From CCW Limit Input: Jumper 16-19, Remove 17-19,18-19

(Optional) From CW Limit Input: Jumper 18-19, Remove 16-19,17-19

See Figure 6-1 for jumper locations.

When the Home command is issued, the standard direction of motor rotation is CCW. If CW rotation is required, jumpers 23-24-25 on the DM4005 and DM4001 board must be reconfigured.

The jumper definition follows:

(Standard) CCW Home - Jumper 23-24, Remove 24-25

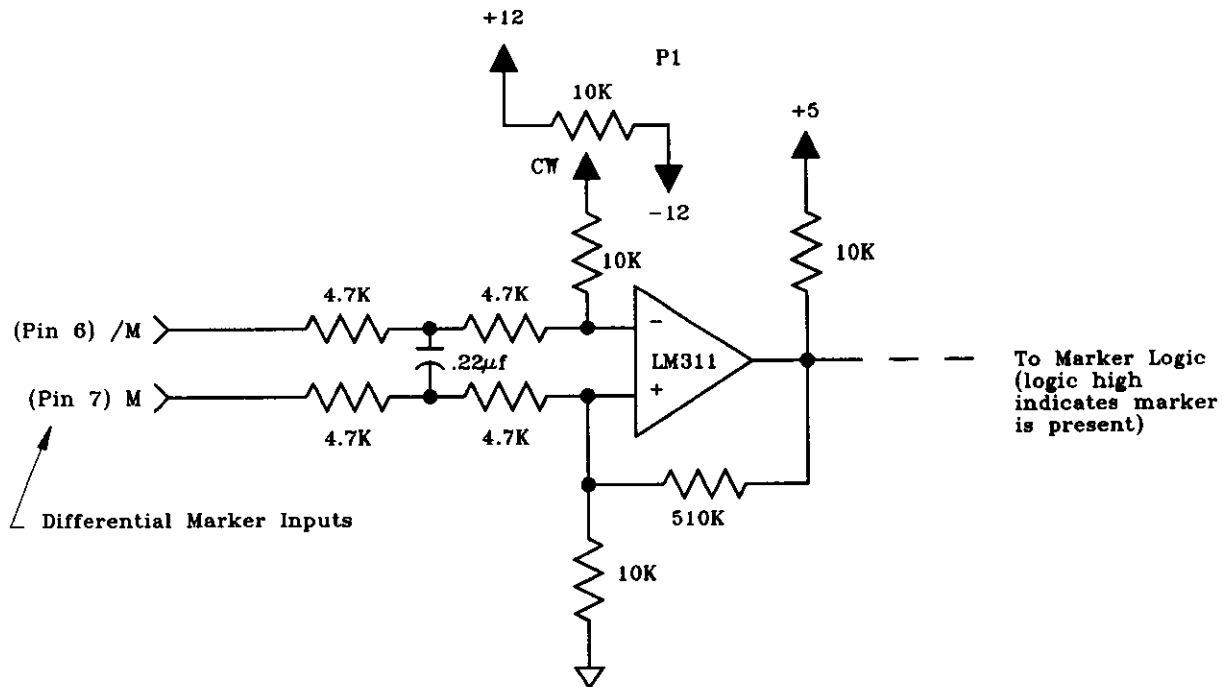
(Optional) CW Home - Jumper 24-25, Remove 23-24

See Figure 6-1 for jumper locations.

NOTE: Positive move commands to the Unidex 14 turns the motor CCW (when looking into the flange).

6-1-1-3: OUTLINE OF MARKER BUFFER CIRCUIT:

Shown below is a circuit diagram of the differential Input Marker circuit:



Potentiometer P1 (see Figure 6-1) in the circuit diagram above, allows the threshold of the Marker input signal to be adjusted. The threshold level is increased by turning P1 CW. To activate the Marker logic, the voltage signal at "M" must rise slightly higher than the voltage signal at "/M", plus the threshold setting at P1. In other words:

$$M > /M + P1 (\text{threshold})$$

For a TTL Marker connection, the signal common of the TTL circuit should be tied to signal common and the Marker tied to "M". P1 should be adjusted to +1VDC at the P1 wiper.

NOTE: P1 will be factory set such that proper "Home Cycle" operation will occur when the Marker option is absent or when interfacing to an Aerotech stage/motor with a "Home Marker" (HM) option. This setting is approximately -1VDC at P1 wiper.

6-1-1-4: LIMIT SWITCH POLARITY SELECTION

As a standard, Unidex 14 Controllers are configured to interface to normally open (active low) limit switches (CCW, CW and Home). If use of normally closed (active high) limit switches is required, jumpers 1 through 6 and 13 through 15 on the DM4005 and DM4001 board must be reconfigured.

The jumper definitions follow:

- (Standard) CCW Limit, Normally Open - Jumper 1-2, Remove 2-3
- (Optional) CCW Limit, Normally Closed - Jumper 2-3, Remove 1-2
- (Standard) CW Limit, Normally Open - Jumper 5-6, Remove 4-5
- (Optional) CW Limit, Normally Closed - Jumper 4-5, Remove 5-6
- (Standard) Home Limit, Normally Open - Jumper 13-14, Remove 14-15
- (Optional) Home Limit, Normally Closed - Jumper 14-15, Remove 13- 14

See Figure 6-1 for jumper locations.

6-1-1-5: +/- DIRECTION DEFINITION AND OPTIONS

When a "+" direction is programmed into a Unidex 14, the motor will rotate in the CCW direction.

NOTE: CW and CCW rotational references are viewed as "looking into" the motor mounting flange.

When a "-" direction is programmed, the motor will rotate in the CW direction. If the opposite convention is required, jumpers 20-21-22 may be reconfigured.

The jumper definition follows:

- (Standard) "+" = CCW, "-" = CW - Jumper 21-22, Remove 20-21
- (Optional) "-" = CCW, "+" = CW - Jumper 20-21, Remove 21-22

See Figure 6-1 for jumper locations.

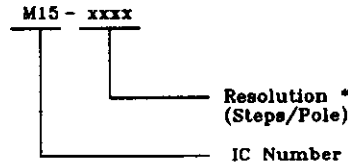
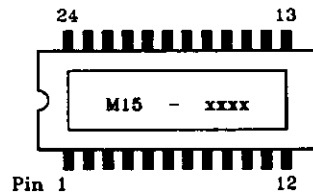
6-1-1-6: SELECTING STEPPING MOTOR RESOLUTION

The DM4005 and DM4001 series can be programmed to drive a Stepping motor at 250 different Stepping resolutions. As was described in part (6-1-1-1 (circuit description), a Stepping resolution range can be selected between 200 Steps/Rev and 50,000 Steps/Rev inclusive, in increments of 200 steps (i.e., 200, 400, 600,....49600, 49800 and 50000 Steps/Rev).

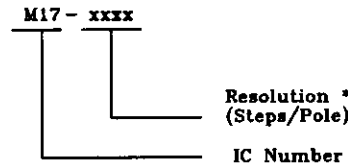
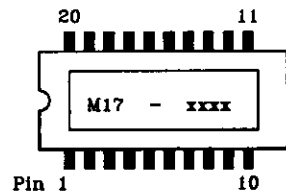
The changing of resolution on the DM4005 and DM4001 involves the programming of two IC chips, M15 and M17. This programming cannot be done in the field. However, M15 and M17 are easily changed in the field since both ICs are mounted on IC sockets.

The part numbering system for M15 and M17 is shown below. These part numbers must be used when ordering different control resolutions.

For IC M15:



For IC M17:



***NOTE:** This number defines the resolution (Steps/Pole)

For example:

POLE RESOLUTIONS
(STEPS/POLE)

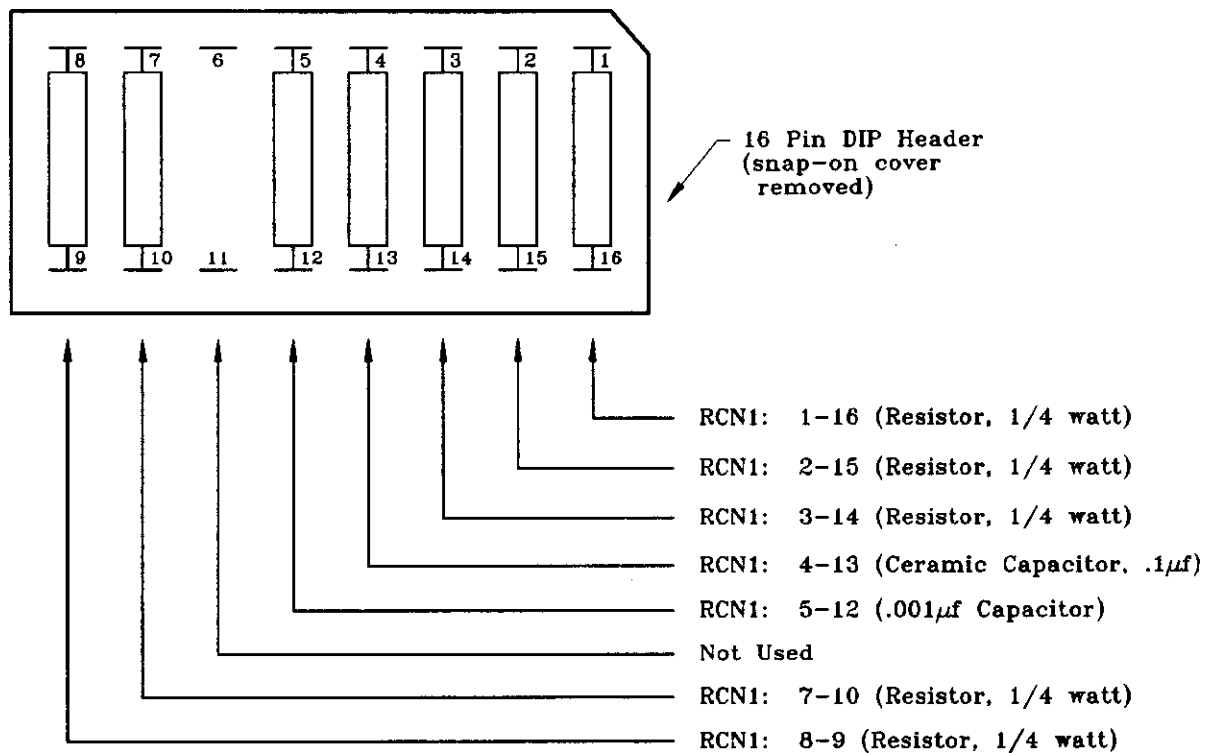
1000	→	50000 Steps/Rev	For standard motors (1.8° /full step)
720	→	36000 Steps/Rev	
80	→	4000 Steps/Rev	
16	→	800 Steps/Rev	

The Step/Rev numbers listed above apply to a 50-pole, 1.8 degree per full step Stepping motor. When determining Step/Rev for other multiple pole motors, multiply the given steps/pole resolution by the number of poles for the given Stepping motor. The resulting number will then be the Step/Rev resolution for that motor.

When changing resolutions, it is common to change Personality Module RCN1 also, so that Home speed and Slew speed (if applicable) vary according to resolution.

6-1-1-7: PERSONALITY MODULE (RCN1)

The Personality Module is a 16-pin DIP wafer consisting of 5 resistors and 2 capacitors. This module contains all of the parameters used to match a given Stepping Motor with a given resolution for the DM4005 or DM4001. A representation of a Personality Module is shown below (see also Figure 6-1).



The parameter settings of RCN1 are explained in the following subsections.

6-1-1-8: RCN1: 8-9, HOME CLOCK OSCILLATOR ADJUSTMENT

The adjustment for the Oscillator which sends the Stepping Motor into a limit involves RCN1: 8-9. This value can be determined as follows:

$$\text{Home Clock Frequency (Hz)} \sim \frac{1}{1.5(.001 \times 10^{-6}) (\text{RCN1: 8-9})}$$

Where RCN1: 8-9 is in ohms

6-1-1-9: RCN1: 3-14 AND RCN1: 2-15, LO/HI CURRENT LEVELS

The low current level (standby current when motor is at rest) is set by RCN1: 2-15. This value of current can be calculated as follows:

$$\text{Standby Current Level (amps)} = N \frac{1 \times 10^4}{\text{RCN1: 2-15}}$$

Where RCN1: 2-15 is in ohms

N = 1 for DM4001

N = 5 for DM4005

The high current level (running current) involves the selection of RCN1: 3-14 and RCN1: 2-15 (determined above). The value can be calculated as follows:

$$\text{Running Current Level (amps)} = N \frac{1 \times 10^4}{\text{RCN1: 2-15}} + \frac{1 \times 10^4}{\text{RCN1: 3-14}}$$

Where RCN1: 2-15 and RCN1: 3-14 are in ohms

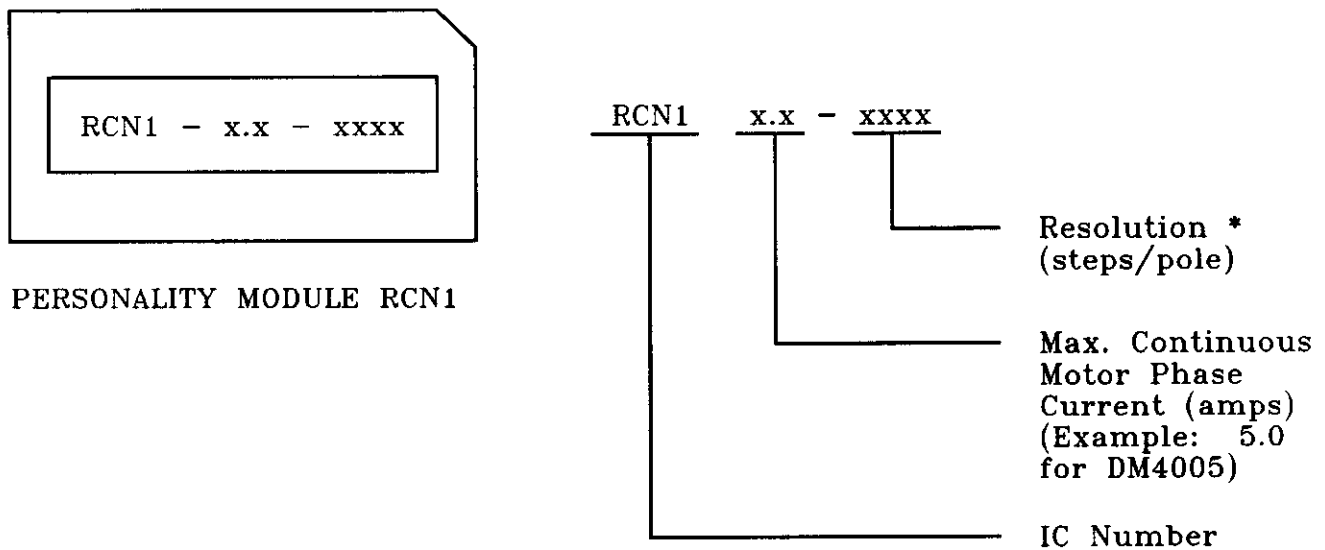
N = 1 for DM4001

N = 5 for DM4005

NOTE: RCN1: 1-16, RCN1: 4-13, RCN1: 6-11 and RCN1: 5-12 are factory selected for the DM4001 and DM4005 Series. These components should never be changed by the user.

The DM4005 and DM4001 are shipped with standard Personality modules, which are defined by the resolution and rated phase current of the Stepping motor being used. If the DM4000 Series is being used with a standard Aerotech Stepping motor, the user need only define the desired resolution in which the motor is to operate. If the DM4000 Series is not being used with a standard Aerotech motor, the user must define the type of Stepping motor being used, with its rated phase current, as well as the DM4000 Series resolution in which it is to operate.

A list of Personality modules used with standard Aerotech stepping motors is provided following the next illustration. A label on the Personality modules defines the resolution (of the internal oscillators) and motor phase current parameters for that module. If the user is not using Aerotech motors, a custom Personality module may be ordered using the part numbering format shown below. Given the rated current and resolution information for a particular motor, Aerotech will set the Slew oscillators, the Home oscillator and the Run/Standby phase current levels. Although it is not generally recommended, the user also has the option of changing the parameters of the Personality module using the equations listed in Section 6-1-1-8, and 6-1-1-9.



* **NOTE:** This number defines the resolution (in steps per pole), for which the slew and Home oscillator, as well as the current levels of the Personality module, are "tailored" to operate.

Recommended standard RCN1 modules for the DM4000 Series are:

Aerotech Translator	Motor	Phase Current (amps)	Torque (oz-in)	Recommended Personality Module (RCN1)
DM4001	50SM	1	38	RCN1 - 1.0 - xxxx
DM4005	101SM	4.6	90	RCN1 - 5.0 - xxxx

Contact Aerotech Inc. for Customized Personality Modules

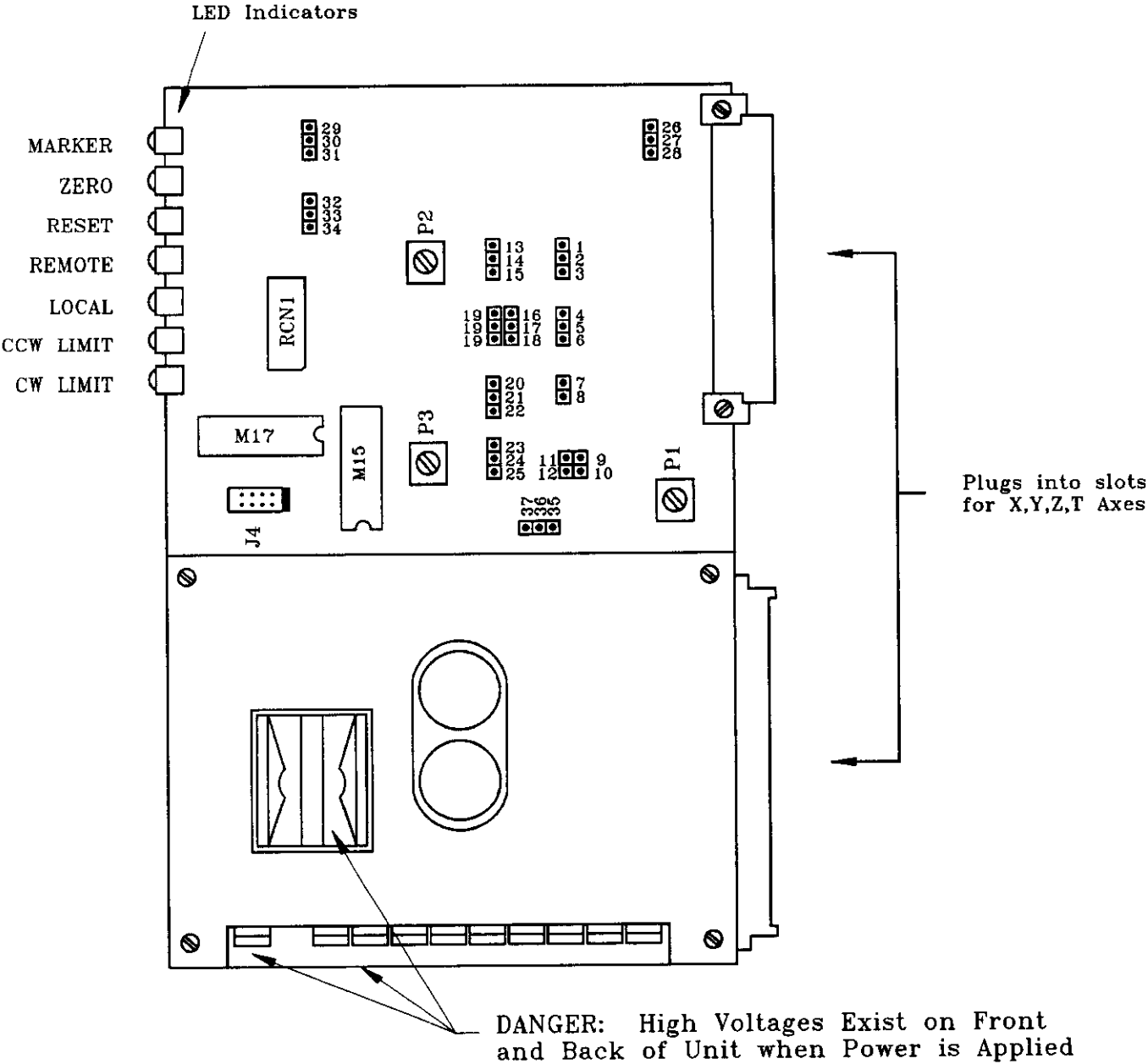


Figure 6-1: Outline of DM4005 and DM4001 Stepping Drive Module

STEPPING DRIVE P/N	DM4001	DM4005
STEPPING MOTOR P/N	50SM	101SM
STATIC TORQUE OZ.-IN.	38	90
MOTOR SPEED MAX. RPM	1000	1875
MAXIMUM MOTOR OUTPUT POWER WATTS	12	53
MOTOR ROTOR INERTIA OZ.-IN.-SEC. ² Kg-M ²	.166 0,0118 x 10 ⁻³	.5 0,035x10 ⁻³
MOTOR FRAME NEMA 2	23	23
MOTOR TYPE (AMPS/PHASE)		
STEPPING DRIVE VOLTS DC AMPS TYPE	40 1 UNIPOLAR	40 5 UNIPOLAR
WEIGHTS MOTOR (LBS.) (KG) MOTOR/DRIVE (LBS.) (KG)	2.3 1.05 12.00 5.45	3.6 1.64 14.00 6.36
INPUT POWER VOLTS (50/60 HZ) AMPS Allowable voltage tolerance (+/- 10% Max.)	115/230 .5	115/230 1

Table 6-1: Stepping Drive Specifications

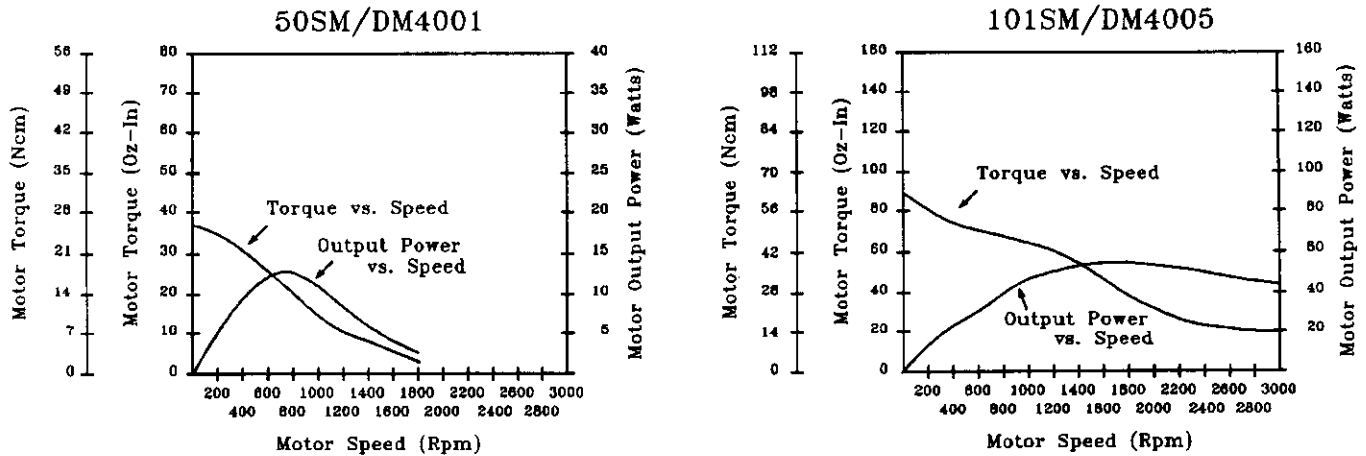


Figure 6-2: DM4001, DM4005, DM1501 Torque/Speed Curves

6-1-2: DMV8008 AND DMV16008 (Model U14H only) DRIVE MODULES

6-1-2-1: CIRCUIT DESCRIPTION

The DMV8008 and DMV16008 are designed to accurately control a standard bipolar Stepping motor by utilizing a unique sin/cos translator control. This control is capable of dividing a full-step size into 250, evenly spaced micro steps. For a 50 pole (200 full step) (standard) Stepping motor, this subdivision of a full step provides electrical resolutions of up to 50,000 steps per revolution. This sin/cos translator control, as mentioned in the introduction, can be easily altered to provide other step sizes. By simply programming two IC chips, a host of other incremental step sizes can be chosen, ranging from 200 steps/rev (full step) up to 50,000 steps/rev in 200 step intervals. In other words, there are 250 choices of stepping resolutions.

The DMV Series Stepping drive modules require a DC bus power supply and +5, ± 12 VDC control supply for operation. The external DC bus power supply, made up mainly of Transformer T1 and/or T2 and capacitors C1 through C4 are shown in Figures 5-2 and 5-3. A separate power supply board module (explained later in this Chapter) is required for +5, ± 12 VDC control power to the DMV Series module.

The DMV8008 can be used in Unidex 14 chassis U14S and U14R and U14H. The DMV 16008 is limited to use in the U14H chassis only. Note that in the case of the DMV16008, Transformers T1 and T2 are not required if the incoming supply to the U14H chassis is 115VAC. The DMV16008 is operating "off line" in this particular case.

An outline of the DMV8008 and DMV16008 drive modules is shown in Figure 6-3. Table 6-2 provides Specifications for the DMV8008 and the DMV16008. Speed/Torque Curves are provided in Figure 6-4.

6-1-2-2: "HOME" REFERENCE DEFINITION & OPTIONS

All Unidex 14 Controllers are capable of generating a cold start reference position, which is the "Home" position. The basic Home cycle involves the following series of events. When the "Go Home" command has been issued, the motor will turn CCW (standard) or CW (optional) until a "Home Limit Switch" activation occurs.

NOTE: The CCW and CW rotational references are viewed "looking into" the motor mounting flange.

Upon "Home Limit Switch" activation, the motor will reverse and rotate in the opposite direction until the switch deactivates. If no "Marker" pulse option exists, upon switch deactivation, the motor will reverse and stop. If a "Marker" pulse option does exist (see Section 6-1-2-3), the motor will reverse and continue to rotate until the "Marker" pulse is located and then stops.

The speed at which the Home cycle occurs is factory set at 120 RPM. A low home speed is used for achieving an accurate home reference position. If a different speed is required, see Section 6-1-2-7 for appropriate Personality module changes.

For most rotary motion stages, the "Home Limit Switch" referenced above is an independent switch incorporated specifically for the "Home" cycle. For linear motion stages, the "Home Limit Switch" could be an independent switch as well. However, in most cases, the CCW or CW limit switches perform "double duty" and act as the "Home Limit Switch". The "Home Limit Switch" wiring either to the CW or CCW limit switch is done at the motor/stage external to the Unidex 14.

If it is not possible to use the "Home Limit" input at J13, 14, 19 or 20 such as when Opto 22 coupled limits are being used, the "CCW" or "CW limit" input may be paralleled to the "Home Limit" input by reconfiguring jumpers JP2-1 through 6 on the DMV16008/DMV8008 PC board. Note that this scheme is possible only with linear stages and not with rotary stages.

The jumper definition follows:

HOME LIMIT SOURCE:

- (Standard) From Home Limit Input: Jumper JP2-3 to 4 , Remove JP2- 1 to 2, JP2-5 to 6.
- (Optional) From CCW Limit Input: Jumper JP2-1 to 2, Remove JP2-3 to 4, JP2-5 to 6.
- (Optional) From CW Limit Input: Jumper JP2-5 to 6, Remove JP2-1 to 2, JP2-3 to 4.

See Figure 6-3 for jumper locations.

When the Home command is issued, the standard direction of motor rotation is CCW. If CW rotation is required, jumpers JP8-1 through 3 on the DMV16008/DMV8008 board must be reconfigured.

The jumper definition follows:

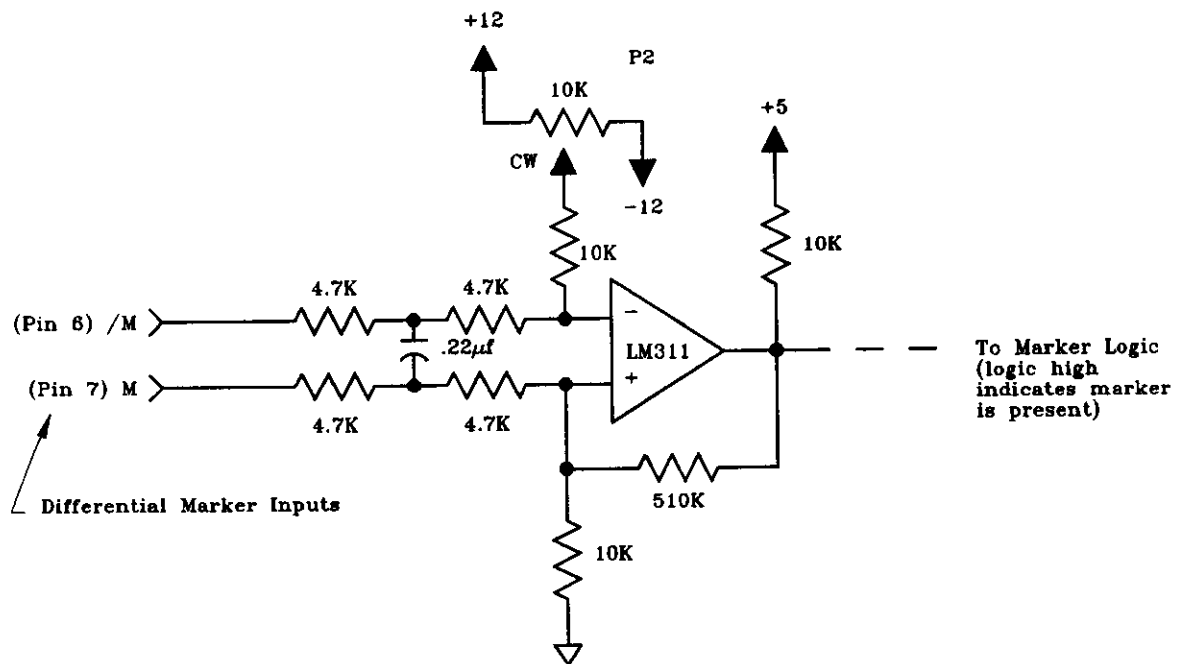
(Standard) CCW Home - Jumper JP8-1 to 2, Remove JP8-2 to 3.

(Optional) CW Home - Jumper JP8-2 to 3, Remove JP8-1 to 2.

See Figure 6-3 for jumper locations.

6-1-2-3: OUTLINE OF MARKER BUFFER CIRCUIT

Shown below is a circuit diagram of the differential input marker circuit:



Potentiometer P2 (see Figure 6-3) in the previous circuit diagram, allows the threshold of the Marker input signal to be adjusted. The threshold level is increased by turning P2 CW. To activate the Marker logic, the voltage signal at "M" must rise slightly higher than the voltage signal at "/M", plus the threshold setting at P2. In other words:

$$M > /M + P2 \text{ (threshold)}$$

For a TTL Marker connection, the signal common of the TTL circuit should be tied to signal common with the Marker tied to "M". P2 should be adjusted to approximately +1VDC at the P2 wiper.

NOTE: P2 will be factory set so that proper "Home Cycle" operation will occur when the marker option is absent, or when interfacing to an Aerotech stage/motor with a "Home Marker" (HM) option. This setting is approximately -1VDC at the P2 wiper.

6-1-2-4: LIMIT SWITCH POLARITY SELECTION

As a standard, Unidex 14 controllers are configured to interface to normally open (active low) limit switches (CCW, CW and Home). If use of normally closed (active high) limit switches is required, jumpers JP3-1 through 3, JP4-1 through 3, and JP5-1 through 3 on the DMV16008/DMV8008 board must be reconfigured.

The jumper definitions follow:

- (Standard) CCW Limit, Normally Open - Jumper JP3-1 to 2, Remove JP3-2 to 3.
- (Optional) CCW Limit, Normally Closed - Jumper JP3-2 to 3, Remove JP3-1 to 2.
- (Standard) CW Limit, Normally Open - Jumper JP4-1 to 2, Remove JP4-2 to 3.
- (Optional) CW Limit, Normally Closed - Jumper JP4-2 to 3, Remove JP4-1 to 2.
- (Standard) Home Limit, Normally Open - Jumper JP5-1 to 2, Remove JP5-2 to 3.
- (Optional) Home Limit, Normally Closed - Jumper JP5-2 to 3, Remove JP5-1 to 2.

See Figure 6-3 for jumper locations.

6-1-2-5: +/- DIRECTION DEFINITION & OPTIONS

When a "+" direction is programmed into a Unidex 14, the motor will rotate in the CCW direction.

NOTE: CCW rotation is viewed as "looking into" the motor mounting flange.

When a "-" direction is programmed, the motor will rotate in the CW direction. If the opposite convention is required, jumpers JP7-1 through 3 may be reconfigured.

The jumper definition follows:

(Standard) "+" = CCW, "-" = CW - Jumper JP7-2 to 3, Remove JP7-1 to 2.

(Optional) "-" = CCW, "+" = CW - Jumper JP7-1 to 2, Remove JP7-2 to 3.

See Figure 6-3 for jumper locations.

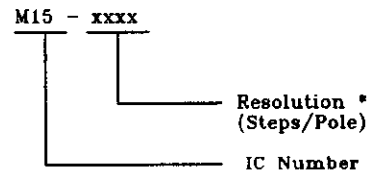
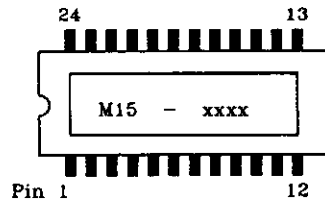
6-1-2-6: SELECTING STEPPING MOTOR RESOLUTION

The DMV Series modules can be programmed to drive a Stepping motor at 250 different stepping resolutions. As was described in Item 6-1-2-1 (*Circuit Description*), a stepping resolution range can be selected between 200 steps/rev and 50,000 steps/rev inclusive, in increments of 200 steps (i.e., 200, 400, 600,, 49600, 49800 and 50000 steps/rev).

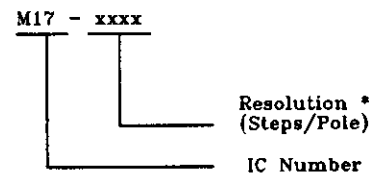
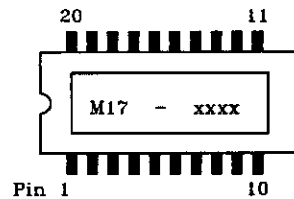
The changing of resolution on the DMV Series involves the programming to two IC chips, M15 and M17 (see Figure 6-3). This programming cannot be done in the field. However, M15 and M17 can be changed very easily in the field since both ICs are mounted on IC sockets.

The part numbering system for M15 and M17 is shown below. These part numbers must be used when ordering different control resolutions.

For IC M15:



For IC M17:



*** NOTE:** This number defines the resolution (in steps/pole).

For example:

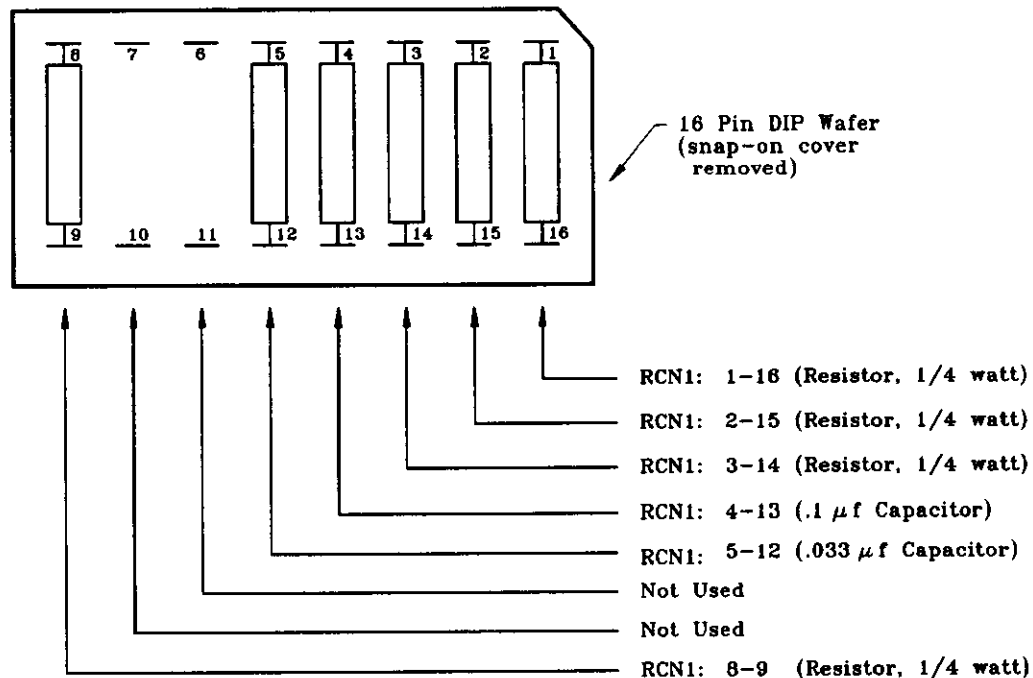
POLE RESOLUTIONS
(STEPS/POLE)

1000	→	50000 Steps/Rev	For standard motors (1.8°/full step)
720	→	36000 Steps/Rev	
80	→	4000 Steps/Rev	
16	→	800 Steps/Rev	

As was mentioned in the introduction to this manual, the step/rev numbers listed above apply to a 50-pole 1.8 degree per full step Stepping motor. When determining step/rev for other multiple pole motors, multiply the given steps/pole resolution by the number of poles for the given Stepping motor. The resulting number will then be the step/rev resolution for that motor.

6-1-2-7: PERSONALITY MODULE (RCN1)

The Personality module is a 16-pin DIP wafer consisting of 4 resistors and 2 capacitors. This module contains all of the parameters used to match a given Stepping motor with a given resolution for the DMV Series. A representation of a Personality module is shown below (see also Figure 6-3):



The parameter settings of RCN1 are explained in the following subsections.

6-1-2-8: RCN1: 8-9, HOME CLOCK OSCILLATOR ADJUSTMENT

The adjustment for the oscillator which sends the Stepping motor into a limit involves RCN1: 8-9. This value can be determined as follows:

$$\text{Home Clock Frequency (Hz)} \sim \frac{1}{.75(.033 \times 10^{-6}) (\text{RCN1: 8-9})}$$

Where RCN1: 8-9 is in ohms

6-1-2-9: RCN1: 3-14 & 2-15, LO/HI CURRENT LEVEL ADJUSTMENT

The RCN1 low current level (standby current) is set by RCN1: 2-15. This value of current can be calculated as follows:

$$\text{Standby Current Level (amps)} = 8 \left[\frac{1 \times 10^4}{\text{RCN1: 2-15}} \right]$$

where RCN1: 2-15 is in ohms

The high current level (running current) involves the selection of RCN1: 3-14 and RCN1: 2-15 (determined above). The value can be calculated as follows:

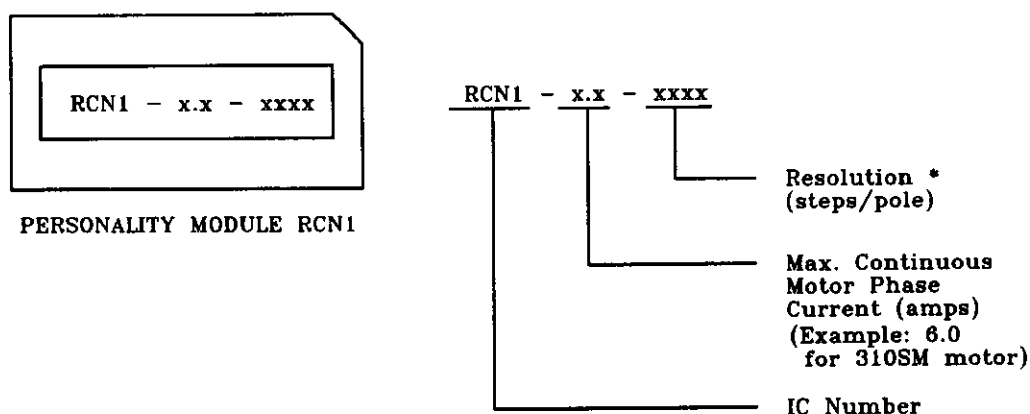
$$\text{Running Current Level (amps)} = 8 \left[\frac{1 \times 10^4}{\text{RCN1: 2-15}} + \frac{1 \times 10^4}{\text{RCN1: 3-14}} \right]$$

where RCN1: 2-15 and RCN1: 3-14 are in ohms

NOTE: RCN1: 1-16, RCN1: 4-13 and RCN1: 5-12 are factory selected for the DMV Series modules. These components should **never** be changed by the user.

The DMV Series modules are shipped with a standard Personality module, which is defined by the resolution and rated phase current of the Stepping motor being used. If the DMV Series is being used with a standard Aerotech Stepping motor, the user need only define the desired resolution in which the motor is to operate. If the DMV Series is not being used with a standard Aerotech motor, the user must define the type of Stepping motor being used, with its rated phase current, as well as the DMV Series resolution in which it is to operate.

A list of Personality modules used with standard Aerotech Stepping motors is listed below. The label on the Personality module defines the resolution (of the internal oscillators) and motor phase current parameters for that module. If the user is not using Aerotech motors, a custom Personality module can be ordered using the part numbering format shown in the following illustration. Given the rated current and resolution information for a particular motor, Aerotech will set the Home oscillator and the Run/Standby phase current levels. Although it is not generally recommended, the user also has the option of changing the parameters of the Personality module by using the equation listed in Items 6-1-2-8 and 6-1-2-9 of this section.



- * This number defines the resolution (in steps per pole), for which the Home oscillator, as well as the current levels of the Personality module, are "tailored" to operate.

Recommended standard RCN1 modules for the DMV Series modules:

Aerotech Translator	Aerotech Motor	Phase Current (amps)	Torque (oz-in)	Recommended Personality Module (RCN1)
DMV8008	310SM	6	370	RCN1 6.0 - xxxx
DMV16008	1010SM	8.6	1050	RCN1 8.0 - xxxx

Contact Aerotech for Customized Personality Modules.

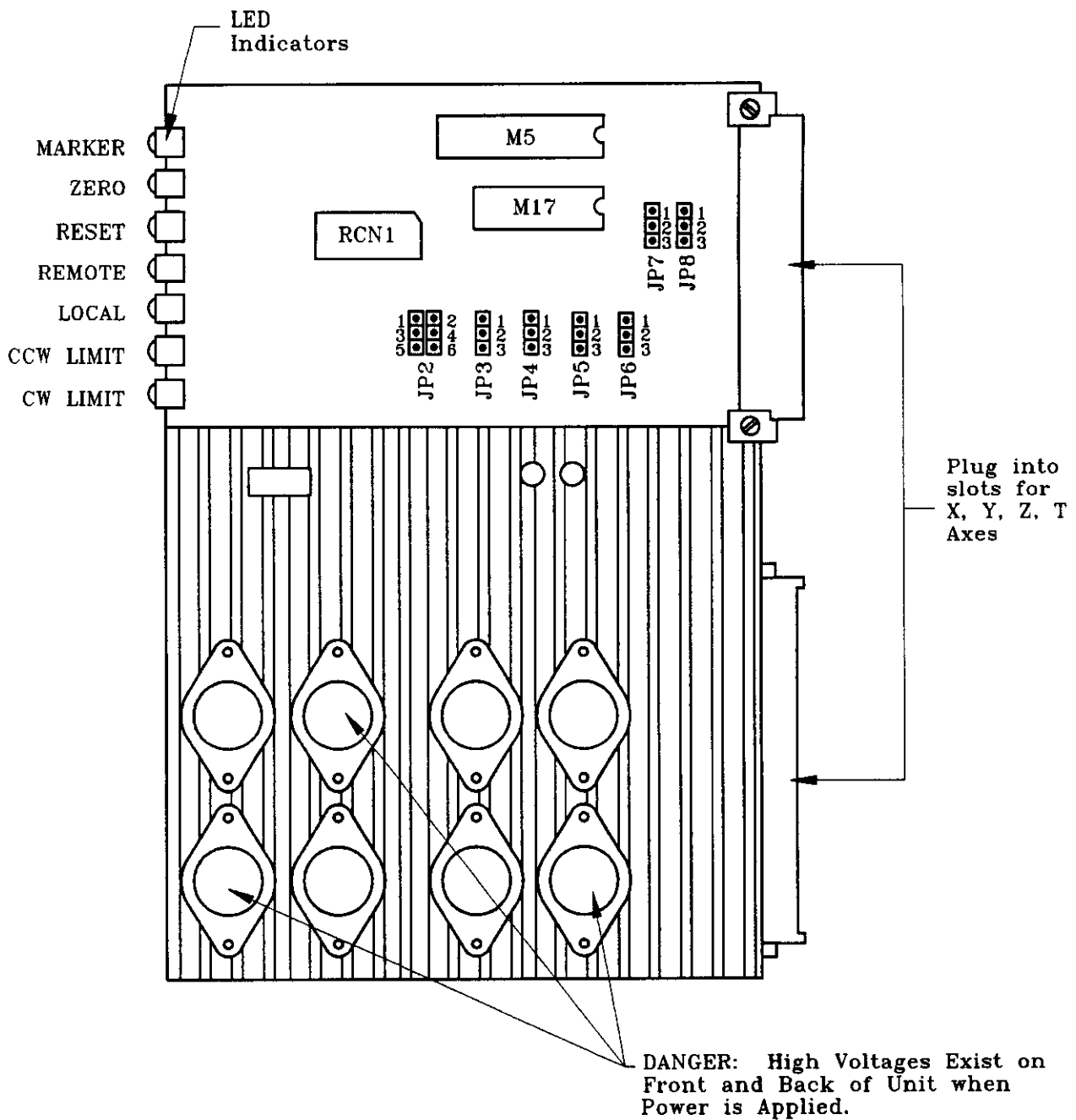


Figure 6-3: Outline of the DMV8008 and DMV16008 Stepping Drive Modules

STEPPING DRIVE P/N	DMV8008	DMV16008
STEPPING MOTOR P/N	310SM	1010SM
STATIC TORQUE OZ.-IN. N-m	370 2,61	1050 7,41
MOTOR SPEED MAX. RPM	2000	2000
MAXIMUM MOTOR OUTPUT POWER WATTS	250	400
MOTOR ROTOR INERTIA OZ.-IN.-SEC. ² Kg-M ²	.027 0,187 x 10 ⁻³	.114 0,805x10 ⁻³
MOTOR FRAME NEMA 2	34	42
MOTOR TYPE (AMPS/PHASE)	6	8
STEPPING DRIVE VOLTS DC AMPS TYPE	80 6 BIPOLAR	160 8 BIPOLAR
WEIGHTS MOTOR (LBS.) (KG)	8.1 3,68	20 9,09
CHASSIS WEIGHTS U14S (LBS.) (KG)	36 16,5	36 16,5
U14R (LBS.) (KG)	30 13,5	30 13,5
U14H (LBS.) (KG)	35 16	35 16
INPUT POWER VOLTS AMPS Allowable voltage tolerance (+/- 10% Max.)	80 VDC 8	160 VDC 8

Table 6-2: DMV8008 and DMV16008 Specifications

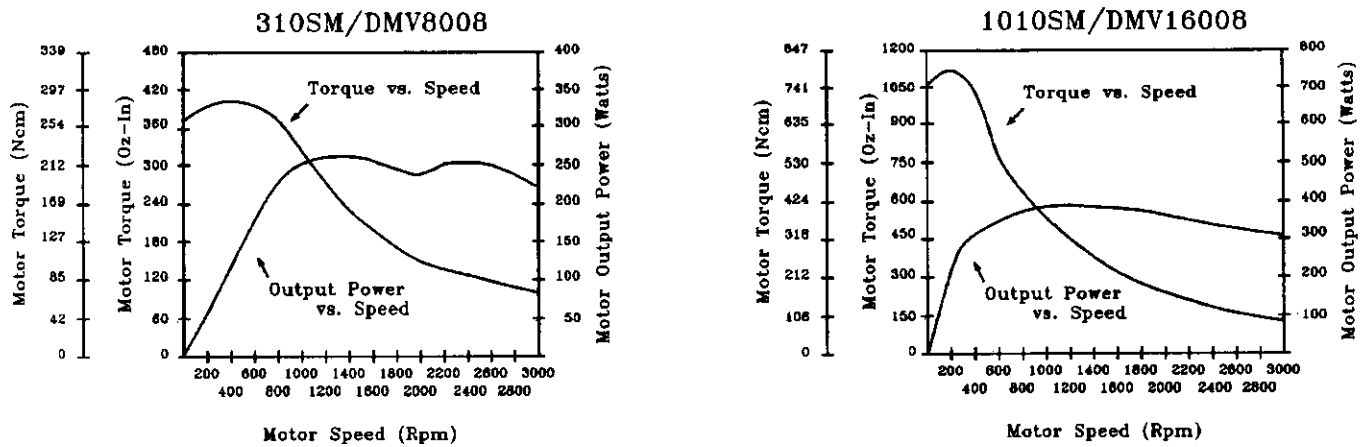


Figure 6-4: DMV8008 and DMV16008 Torque/Speed Curves

SECTION 6-2 DSL3015, 4020, 8020, AND 16020 DC SERVO DRIVE MODULES

6-2-1: CIRCUIT DESCRIPTION

The DSL3015, 4020, 8020 and 16020 DC Servo Drive Modules have been designed to accurately control a standard DC brush Servo motor with torque ranges of between 10 and 400 oz-in. This module runs the Servo motor under closed loop control through an incremental square wave or amplified sine wave Encoder. Position and velocity loops are both "closed" with circuitry contained on this module. Analog "position locking" can be obtained with the use of an amplified sine wave Encoder for applications requiring high "in-position" stability at varying load torques. Protection is included on this module for detecting "faulty" (or missing) Encoder feedback signals or excessive position command to position feedback error. Emergency shutdown is automatic under these conditions.

Unlike the D and DM Series Stepping motor Drive modules described in the previous sections, the DSL DC Servo Drive modules require a DC bus power supply and +5, ± 12 VDC control supply for operation. The external DC bus power supply, made up mainly of Transformer T1 and/or T2 and capacitors C1 through C4 is shown in Figures 5-2 and 5-3. A separate power supply board module (explained later in this chapter) is required for +5, ± 12 VDC control power to the DSL modules.

6-2-2: DSL DC SERVO DRIVE MODULES OPERATION

The DSL position control loop circuit works on the principle of command and feedback pulse summation. That is, CW and CCW command pulses are received from the control board module (described in the next section) and "summed" with the CW and CCW feedback pulses derived from the incremental Encoder. The summation technique involves decoding logic, a counter to sum the feedback and command pulses and a D/A converter (Digital to Analog) whose output represents the position error between the summation of the command and feedback pulses. The position error signal is used as a velocity command signal to the pre-amplifier (described later). In addition to the position control circuit, (described above), an additional circuit is provided to "measure" the rate of CW and CCW feedback pulses per unit time in order to derive a representation for motor speed. The output of this circuit can be said to be an "electronically derived" velocity feedback signal from the motor. This circuit can also be configured to accept an optional tachometer.

For "locking" the motor at rest (zero count position error), a third circuit is provided to "hold" the DC motor firmly within a $\pm 1/2$ step rest. This function is termed Analog Lock, and applies to sine wave type Encoders only. The Analog Lock feature eliminates "one bit" jitter when the motor is in the rest position and an external torque is being applied to its shaft.

A fourth circuit is provided on this board to "sum" the output signals of the three circuits just described. This circuit, labeled the "pre-amplifier" circuit, provides a signal as its output that is representative of the motor torque required to satisfy the position loop. This signal is actually the current command signal to the power amplifier portion of the DSL Drive Module (current drive directly proportional to torque). A block diagram of the four circuits described above is shown on Figure 6-5.

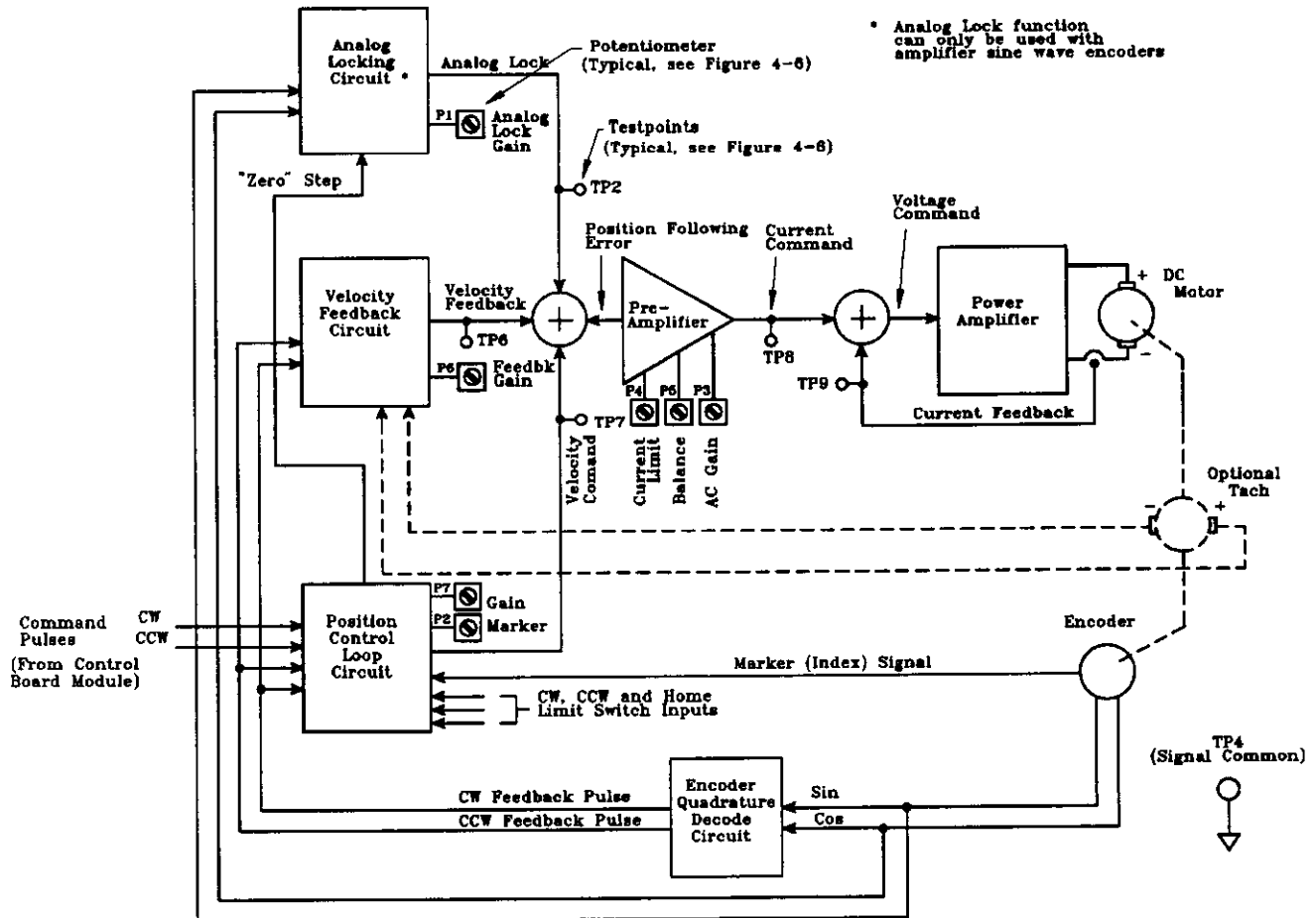


Figure 6-5: Block Diagram of DSL Drive Modules Control Circuit

6-2-3: OPTIONAL TACHOMETER

For the DSL4020, 8020 or 16020 to be used with an optional tachometer, it is necessary to reconfigure jumper JP11 and to remove .47 μ f lead capacitor at RCN5: 3-14.

JP11	RCN5	
	3-14	
1-2	.47 μ f	Electronic Tach (standard)
2-3	Out	Analog Tach (optional)

6-2-4: MOTOR/ENCODER/TACHOMETER PHASING

If the Motor, Encoder, and/or optional Tachometer is user supplied, it must be phased properly.

IMPORTANT: Improper phasing can cause loop instability or even a runaway condition to occur.

Aerotech defines proper Motor, Tach, and Encoder phasing with respect to CW motor rotation when viewed from the motor mounting flange. When a positive voltage is applied to the positive motor lead and a negative voltage to the negative lead, the motor will rotate CW. When the motor turns CW, this should cause the cosine to lead the sine Encoder signal and the Tach to generate a positive feedback voltage.

6-2-5: MOTOR LOAD FUSE RATINGS FOR FUSE F1/CUR LIM

The motor load fuse is located on the bottom of the DSL Servo module (see Figures 6-8 and 6-9). For Aerotech motors, the following fuse rating should be used.

Aerotech Motor	Motor Fuse F1	Current Limit
1017LT	4 amps	± 16 amps
1035LT	4 amps	± 16 amps
1050LT	5 amps	± 20 amps
1075LT	5 amps	± 20 amps
1135LT	5 amps	± 20 amps
1410LT	8 amps	± 20 amps
1960LT	12 amps	± 20 amps

NOTE: All fuses are 3AG type/slow-blow.

The motor fuse is sized according to the continuous current or torque rating of the motor and protects the motor from over-heating. The current limit setting of the DSL Servo modules are meant to limit peak current and is generally set to ± 20 amps or four times the value of the motor fuse, whichever is less.

CAUTION: On smaller motors, due to the thermal time constant, the current limit setting is set at a much lower ratio. Check motor data sheets for actual value.

6-2-6: "HOME" REFERENCE DEFINITION & OPTIONS

All Unidex 14 Controllers are capable of generating a cold start reference position, which is the "Home" position. The basic "Home" cycle involves the following series of events. When the "Go Home" command is issued, the motor will turn CCW (standard) or CW (optional) until a "Home Limit Switch" activation occurs.

NOTE: The CCW and CW rotational references are viewed "looking" into the motor mounting flange.

Upon "Home Limit Switch" activation, the motor reverses and rotates in the opposite direction until the switch deactivates. If a "Marker" pulse option does exist (see Section 6-2-7). The motor reverses and continues to rotate until the "Marker" pulse is located and then stops.

The Home speed is determined by the value of the RCN5: 6-11 resistor and will vary, depending on the system resolution.

Rotary Encoder (Steps/Rev)	Linear Encoder (Steps/mm)	Frequency	RCN5 6-11
200	50	2.5kHz	20K
400	100	5.0kHz	10K
1000 & up	250 & up	10.0kHz	5.1K

For most rotary motion stages, the "Home Limit Switch" referenced above is an independent switch incorporated specifically for the "Home" cycle. For linear motion stages, the "Home Limit Switch" could also be an independent switch. However, in most cases, the "CCW" or "CW" limit switches perform double duty and act as the "Home Limit Switch". The "Home Limit Switch" wiring to either the CW or CCW limit switch is done at the motor/stage, external to the Unidex 14.

It is not possible to use the "Home Limit" input at J13, 14, 19 or 20, when Opto 22 coupled limits are being used. When used in this fashion, "CCW" or "CW" limit input may be paralleled to the "Home Limit" input by reconfiguring jumper JP3 on the DSL PC board. The jumper definition is as follows:

JP3 HOME LIMIT SOURCE:

(Standard) From Home Limit Input: Jumper JP3-3 to 4 , Remove JP3- 1 to 2, JP3-5 to 6.

(Optional) From CCW Limit Input: Jumper JP3-1 to 2, Remove JP3-3 to 4, JP3-5 to 6.

(Optional) From CW Limit Input: Jumper JP3-5 to 6, Remove JP3-1 to 2, JP3-3 to 4.

See Figures 6-8 and 6-9 for jumper locations.

When the Home command is issued, the standard direction of motor rotation is CCW. If CW rotation is required, jumper JP1 on the DSL board must be reconfigured.

JP1 HOME DIRECTION JUMPER:

(Standard) CCW Home - Jumper JP1-2 to 3, Remove JP1-1 to 2.

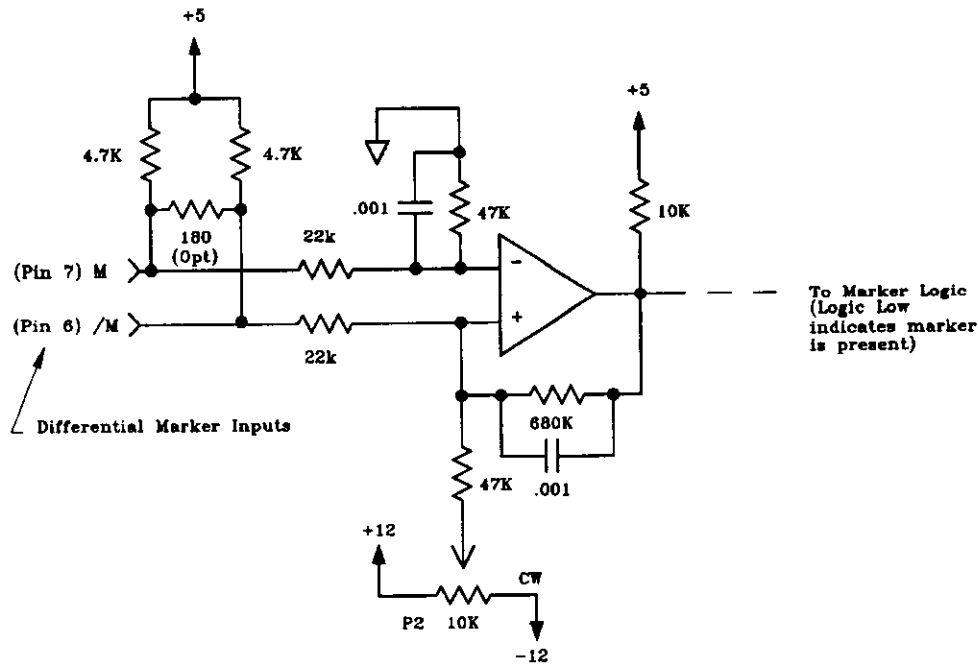
(Optional) CW Home - Jumper JP1-1-2, Remove JP1-2 to 3.

See Figures 6-8 and 6-9 for jumper locations.

NOTE: Positive move commands to the Unidex 14 causes the motor to rotate in the CCW direction.

6-2-7: MARKER INPUT CIRCUIT

Shown below is a circuit diagram of the differential Input Marker circuit:



Potentiometer P2 in the circuit diagram above allows the threshold of the Input Marker signal to be adjusted. The threshold level is increased by turning P2 CCW. To activate the Marker logic, the voltage signal at "M" must rise slightly higher than the voltage signal at "/M", plus the threshold setting at P2. In other words:

$$M /M + P2 (\text{threshold})$$

For a TTL Marker connection, the signal common of the TTL circuit should be tied to signal common with the Marker tied to "M". P2 should be adjusted to approximately +1VDC at the P2 wiper.

6-2-8: LIMIT SWITCH POLARITY SELECTION

As a standard, Unidex 14 controllers are configured to interface to normally open (active low) limit switches (CCW, CW and Home). If use of normally closed (active high) limit switches is required, jumpers JP2, JP4, and JP5 on the DSL board must be reconfigured. The jumper definitions follow:

LIMIT/LIMIT-N JUMPERS (JP2, JP4, JP5)

- (Standard) CCW Limit, Normally Open - Jumper JP5-2 to 3, Remove JP5-1 to 2.
- (Optional) CCW Limit, Normally Closed - Jumper JP5-1 to 2, Remove JP5-2 to 3.
- (Standard) CW Limit, Normally Open - Jumper JP2-2 to 3, Remove JP2-1 to 2.
- (Optional) CW Limit, Normally Closed - Jumper JP2-1 to 2, Remove JP2-2 to 3.
- (Standard) Home Limit, Normally Open - Jumper JP4-2 to 3, Remove JP4-1 to 2.
- (Optional) Home Limit, Normally Closed - Jumper JP4-1 to 2, Remove JP4-2 to 3.

See Figures 6-8 and 6-9 for jumper locations.

6-2-9: ENCODER MULTIPLICATION PARAMETERS

Incremental Encoders produce two output signals generally referred to as sine and cosine. These signals are displaced 90 degrees with respect to each other (i.e., in quadrature with respect to each other). These quadrature signals can be in the form of either amplified sine waves or square waves. (The amplified sine wave type being applicable to analog lock control.)

Through electronic decoding logic on the DSL Servo Module, these quadrature signals can be interpreted as providing 1, 2 or 4 individual steps (machine steps) per one full cycle of sine and cosine.

Multiplying the Encoder signal by 1, 2, or 4 (X1,X2,X4), involves the manipulation of jumpers and, in case of amplified sine wave Encoders, the changing of resistor values on the Encoder interface module (see Figures 6-8 and 6-9). For square wave Encoder operation, only jumpers on the main board must be selected.

ENCODER MULTIPLICATION	MULTIPLIER	R5	R6	R7	R8	JUMPER	JUMPER JP7	JUMPER JP8	JUMPER JP10	JUMPER JP16
SINE WAVE 690C1333	X1	22K	OUT	OUT	43K	2-3	2-3	2-3	2-3	2-3
	X2	22K	OUT	OUT	43K	2-3	2-3	1-2	2-3	2-3
	X4	OUT	43K	22K	OUT	1-2	1-2	2-3	2-3	NONE
SQUARE WAVE 690C1334	X1	(No change)	(No change)	(No change)	(No change)	(No change)	2-3	2-3	2-3	2-3
	X2						2-3	1-2	2-3	2-3
	X4						1-2	2-3	2-3	NONE
Encoder Interface Module							Main Board			

NOTE: Changing resolution will also change system gains. For optimum performance, it may be necessary to change the Accel./Decel. parameter and/or the gain setting of the DSL Servo Module.

6-2-10: ADJUSTMENTS

Seven potentiometers and six test points are provided on the DSL Servo module. A description of these pots and test points are as follows (refer to Figures 6-5, 6-8 and 6-9).

P1

ANALOG LOCK GAIN ADJUST

This pot adjusts the sine lock gain for analog lock in the "zero" step (rest) position. This pot is applicable to sine wave Encoders only. This pot *must* be turned to the full CW position when using square wave type Encoders. Analog lock is adjusted by monitoring testpoint TP2 (with respect to signal common, TP4) and the zero LED indicator. Turning P1 CCW increases the analog lock gain.

Analog lock is adjusted by monitoring testpoint TP2 (with respect to signal common, TP4) and the zero LED indicator. Turning P1 CCW increases the analog lock gain.

When the motor is at rest (no command pulses), the "ZERO" LED indicator must first be set to energize by adjusting the BALANCE pot P5 (discussed below). When the ZERO LED is energized, the analog lock testpoint (TP2) should be monitored (with an oscilloscope) and both the BALANCE pot P5 and the ANALOG LOCK pot P1 should be adjusted until TP2 reads close to zero volts.

P2 MARKER ADJUST

This pot adjusts the threshold of the Marker (index) input pulse from the incremental Encoder (see Marker input circuit, Section 6-5-7).

P3 AC GAIN ADJUST

This pot adjusts the AC gain of the pre-amplifier. Pot P7 and P6 affect the DC as well as the AC gain of the pre-amplifier as discussed below. Turning the AC GAIN pot (P3) CCW increases the AC gain of the pre-amplifier.

P4 CURRENT LIMIT ADJUST

This pot adjusts the clamping level of current applied to the Servo module through the power stage of the DSL Driver. The full CCW position allows maximum current. The full CW position clamps the current command to zero (no motor current). The current command signal can be monitored at test point TP8.

The motor current feedback signal, which is summed with the current command signal to produce a voltage command signal to the power amplifier (see Figure 6-5), may be monitored at testpoint TP9.

Both the Current Command and Current Feedback testpoints (TP8 and TP9 respectively) provide signal gain ratio of ± 3 amps per volt.

The summation circuitry for Current Feedback and Current Command (specifically an integral-lead circuit) provides a motor current bandwidth of approximately 1000 Hz (assuming a 2 - 5 milliHenry load inductance).

P5 BALANCE POT

This pot adjusts analog offsets inherent in the pre-amplifier circuit (see Figure 6-5) which can cause position error. This pot is adjusted when the DC motor is at rest. Set the pot to a position such that the "zero" LED remains lit. Test the netting by making a move and confirming that the "zero" LED lights up when the motor comes to rest.

P6 VELOCITY FEEDBACK GAIN

This pot adjusts the amount of DC Velocity Feedback Gain to the pre-amplifier. Velocity Feedback is electronically derived from the Encoder or from an optional tachometer (see Figure 6-5). Velocity Feedback has two purposes. First, to provide "damping" to the position control loop to eliminate position loop oscillation. Secondly, the Velocity Feedback provides a means of generating position "following error" in the position loop. Following error provides the desirable effect of eliminating position "overshoot" when operating the motor at high Accel./Decel. rates. Testpoint TP6 provides a monitor for the Velocity Feedback signal. Turning this pot CW increases Velocity Feedback Gain and typically reduces motor positioning response.

P7 COMMAND GAIN

This pot adjusts the amount of DC Velocity Command Gain to the pre-amplifier. Command Velocity is derived from the accumulated difference (or error) between the command pulses from the control board module and the feedback pulses generated by the Encoder. Testpoint TP7 provides a monitor for the Velocity Command signal. The ratio of the voltage of this signal with respect to actual command motor

velocity varies with the encoder resolution and the setting of the Velocity Feedback Gain described above, and therefore cannot be directly defined.

NOTE: Adjustment of the COMMAND VELOCITY GAIN pot (P7) and FEEDBACK VELOCITY GAIN pot (P6) may require that the BALANCE pot (P5) be readjusted to obtain the "ZERO" step condition discussed earlier.

6-2-11: ADJUSTING POSITION & VELOCITY LOOP

The DSL Drive Module is factory adjusted for given Aerotech DC Servo motor(s) (i.e., 1050LT, 1135LT, etc.) shipped with the Unidex 14. *If a user-supplied motor and Encoder is to be used with Unidex 14, Aerotech should be notified of the motor and Encoder parameters at the time of purchase to insure compatibility and proper factory set up.*

If the Motor, Encoder and Translation stage are user supplied, the following adjustment procedure is recommended:



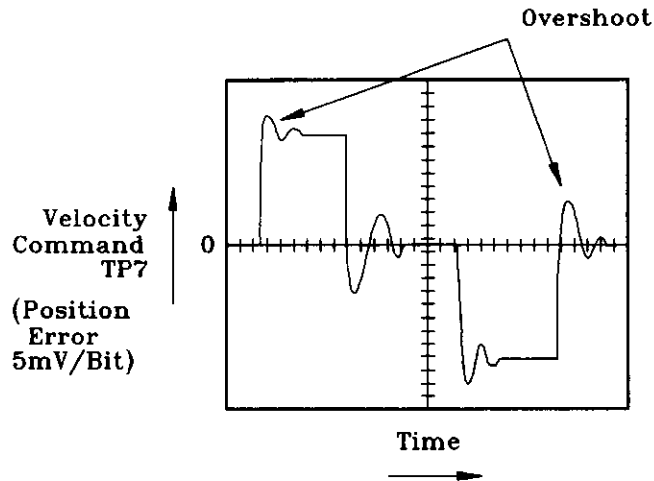
DANGER: PRIOR TO MAKING ANY ELECTRICAL CONNECTIONS OR DISCONNECTIONS. MAKE CERTAIN ALL ELECTRICAL POWER SWITCHES ARE IN THE "OFF" POSITION.

NOTE: An oscilloscope is required for the following procedure.

1. If the Motor, Encoder and optional Tachometer are one assembly, disconnect them from the load if possible.
2. With power turned off, rotate the motor CW (as viewed from the motor mounting flange), and check motor polarity. A positive voltage can be measured from the positive terminal with respect to the negative terminal of the motor.
3. **REMOVE MOTOR FUSE** (see Figures 6-8 and 6-9), connect the oscilloscopes common lead to TP4 (Signal Common), Channel 1 probe to TP3 (Cosine Square) and Channel 2 probe to TP5 (Sine Square). For the placement of testpoints and motor fuse, see Figures 6-8 and 6-9. Apply power, manually turn motor CW and check phasing of Encoder signals at TP3 and TP5. Note that signals at TP3 and TP5 will always be square wave signals, even if sine wave Encoders are used. Also check for a positive Velocity Feedback voltage at TP6 while turning the motor CW.

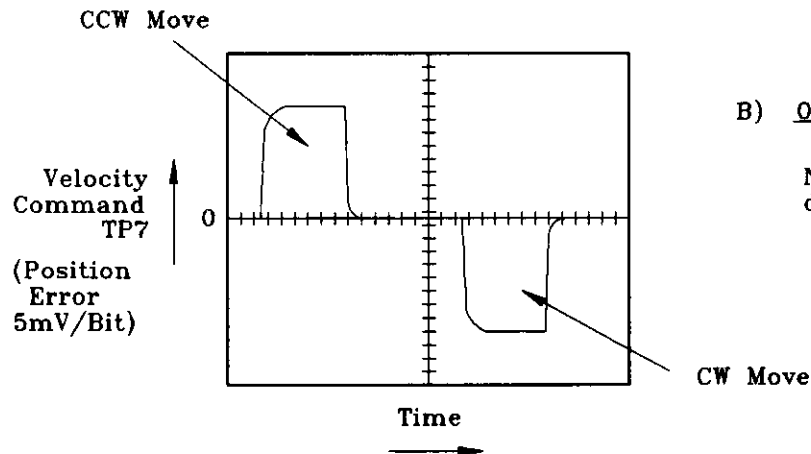
4. Turn off power, replace motor fuse (see Section 6-2-5 for fuse ratings), and turn P1 full CW, P3 full CW, P4 full CW (minimum current limit), P5 mid-range, P6 full CW, and P7 full CCW. This should set the GAIN and CURRENT LIMIT to their lowest adjustable value. Apply power and slowly adjust the CURRENT LIMIT adjustment, P4 CCW and observe the system. If the system is stable, turn P4 full CCW, ± 20 amps, or adjust to desirable level by monitoring TP9, Current Feedback testpoint (3 A/V) while running a program.
5. Write a small program that will run Unidex 14 back and forth with a 1-second dwell between moves. Set the Feedrate for 1/2 of the maximum system speed. Set the Accel./Decel. rate to 10 times the system speed is (steps/sec), linear profiling. Make certain that the move is long enough to affirm that the motor can accelerate and run at programmed speed.
6. Replace the motor fuse, apply power, and run the program detailed in Step 5. Adjust P7 (COMMAND GAIN) CW for less than or equal to $\pm 4V$ of following error at TP7 with respect to TP4 (Signal Common). If the motion stops, this means that the range of the D/A has been exceeded. If this is the case, turn P7 more CW, and turn drive chassis power off, then on again, to clear fault condition and try again. If you do not have enough range with P7, then turn P6 (VELOCITY FEEDBACK) CCW to decrease the following error. If the error condition persists, the problem may be due to one of the following:
 - a. The requested program speed exceeds the capabilities of the motor and amplifier.
 - b. There may not be enough acceleration torque available to accelerate the motor and/or load up to speed. If this is the case, it may be necessary to decrease the Accel./Decel. rate.
7. If the load was disconnected in Step 1, reconnect it.

8. Turn AC GAIN ADJUSTMET, P3, CCW, until the system starts to oscillate. Then turn P3 CW 1/8 turn past the point where it stops oscillating.
9. The system should now be stable and the gain adjustments very conservatively set (over damped). At this point the system should be running the program from Step 5 very smoothly. To optimize positioning time, run the program from Step 5 at maximum system speed and turn P7 (COMMAND GAIN) further CW, to achieve the results at TP7 (VELOCITY COMMAND) as shown in Figure 6-6. If you do not have enough range with the P7 adjustment, it may be necessary to turn P6 (VELOCITY FEEDBACK GAIN) more CCW. However, you must remember that if you adjust P6 you will also have to readjust P3 (AC GAIN) by following the description in Step 8, (or by viewing the results at TP6 while running the program).
10. Increase the Accel./Decel. rate until you start to notice overshoot at TP7 (VELOCITY COMMAND) is noticed.
11. To adjust the Balance for a square wave Encoder system, send the axis to the "Home Position" and adjust P5 (BALANCE) until both the "Marker" and the "Zero" indicator LEDs on the Unidex 14 front panel are energized.
To adjust the Balance for a sine wave Encoder system, adjust the P1 (ANALOG LOCK GAIN) and P5 (BALANCE). When the system is at rest, adjust P5 until the "ZERO" indicator LED for the respective axis is lit on the front panel of the Unidex 14. Connect the probe of the oscilloscope to TP2 (Lock Signal) and the Common lead of the oscilloscope to TP4 (Signal Common). Turn P1 (ANALOG LOCK GAIN) mid-range and adjust P5 (BALANCE) for zero volts ($\pm 0.5V$) at TP2. Execute a one step move and adjust P1 for approximately 25% overshoot at TP2 upon completion of the move with little or no ringing (as shown in Figure 6-7). After adjusting P1 it may also be necessary to readjust P5 once again.



A) Underdamped System

Feedback gain pot P6 turned too far CCW (and/or command gain pot P7 turned too far CW).



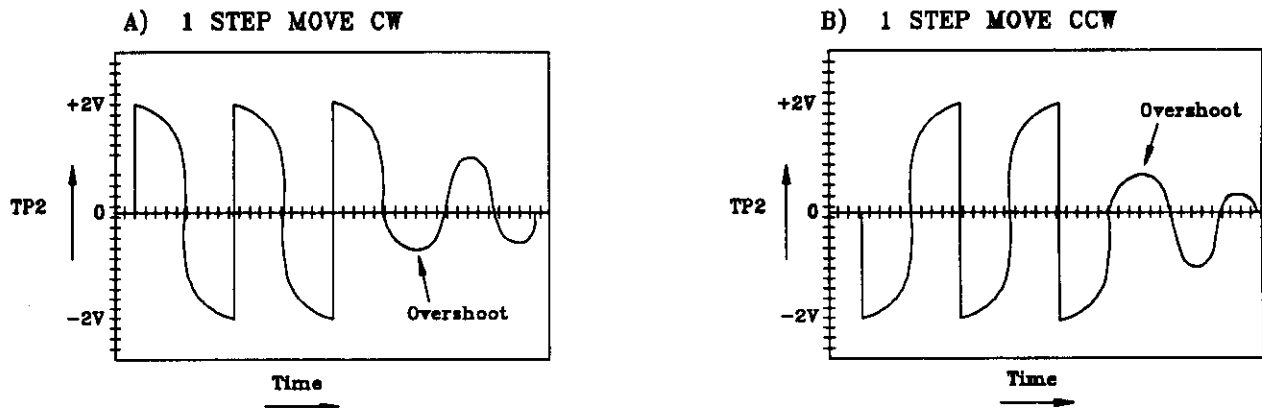
B) Optimally Damped System

No overshoot. Properly damped system.

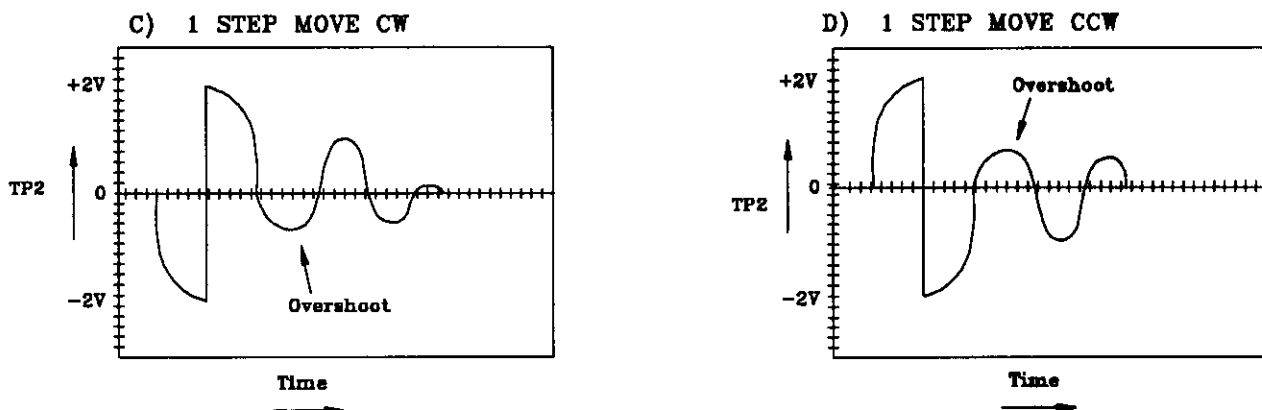
Figure 6-6: Velocity Command refers to TP7 for Underdamped (A), and Optimally Damped (B), Position Loop Response

NOTE: The velocity command shown in Figure 6-6 also represents the position error which is scaled to a 5mV/machine step.

X1 RESOLUTION ANALOG LOCK



X2 RESOLUTION ANALOG LOCK



X4 RESOLUTION ANALOG LOCK

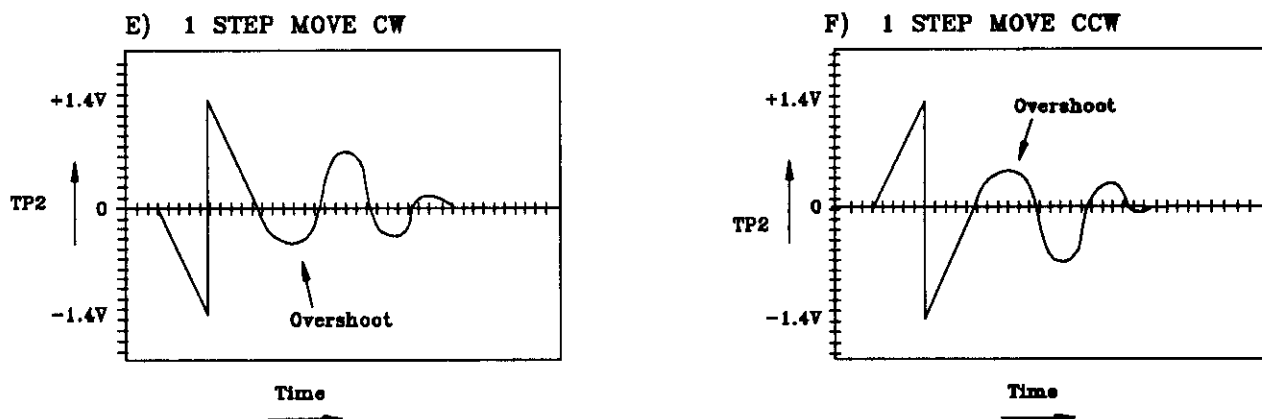


Figure 6-7: Analog Lock Signal at TP2 for X1, X2 and X4 Resolution
(shown for Sine Wave Encoders only)

The DSL Drive Module is Now Ready for Operation

Before programming normal motion, index the motor (at a very low velocity) into the CW Travel Limit switch and CCW Travel Limit switch. Be certain that the CW and CCW limit LEDs energize in their proper sequence and that Unidex 14 responds by stopping motion *before* the motor reaches its mechanical end stop (normally a minimum distance of one motor revolution should exist between the respective limit switch activating point and the mechanical end stop).

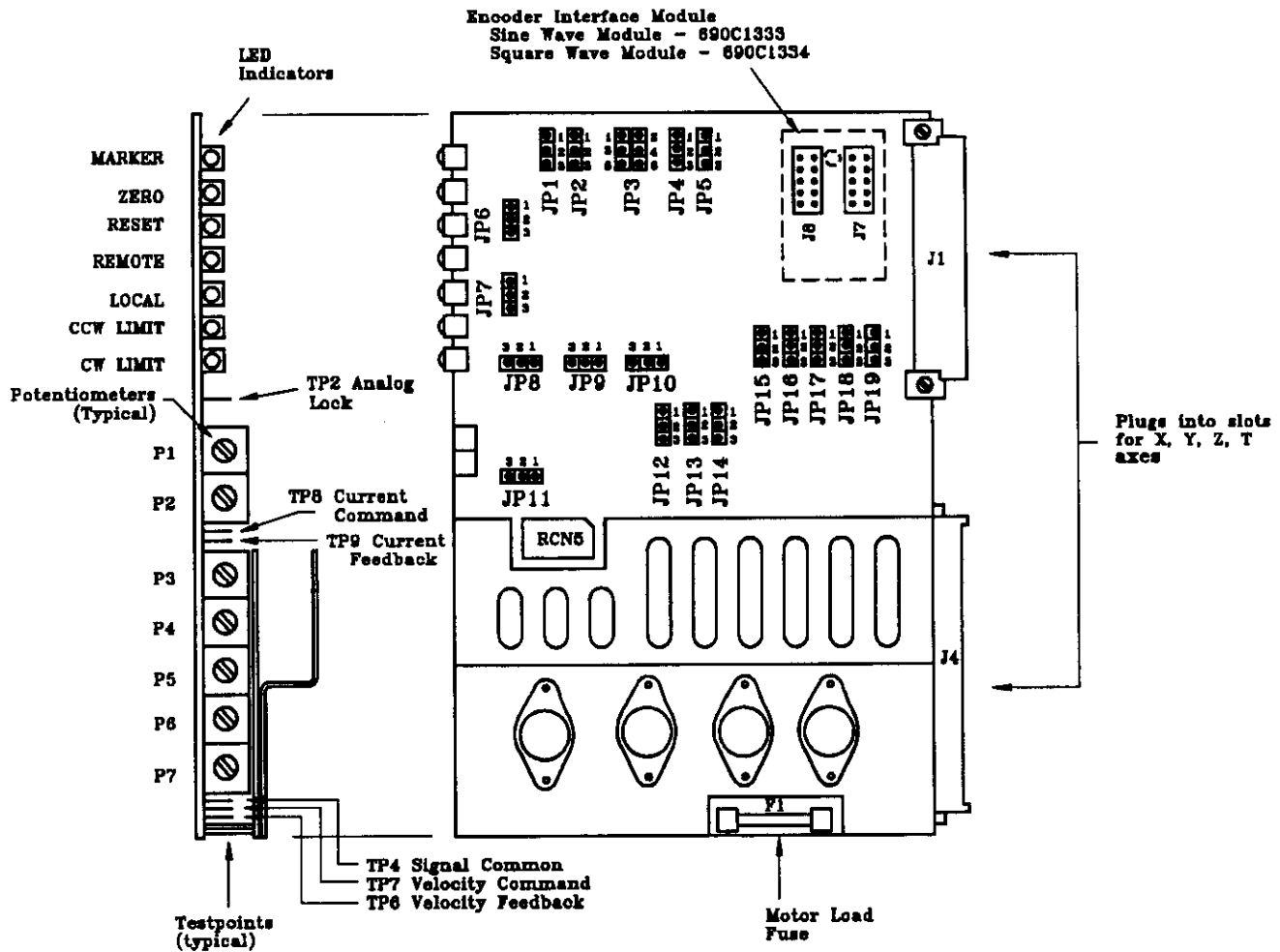


Figure 6-8: Outline of the DSL 4020, 8020 AND 16020DC Servo Drive Module

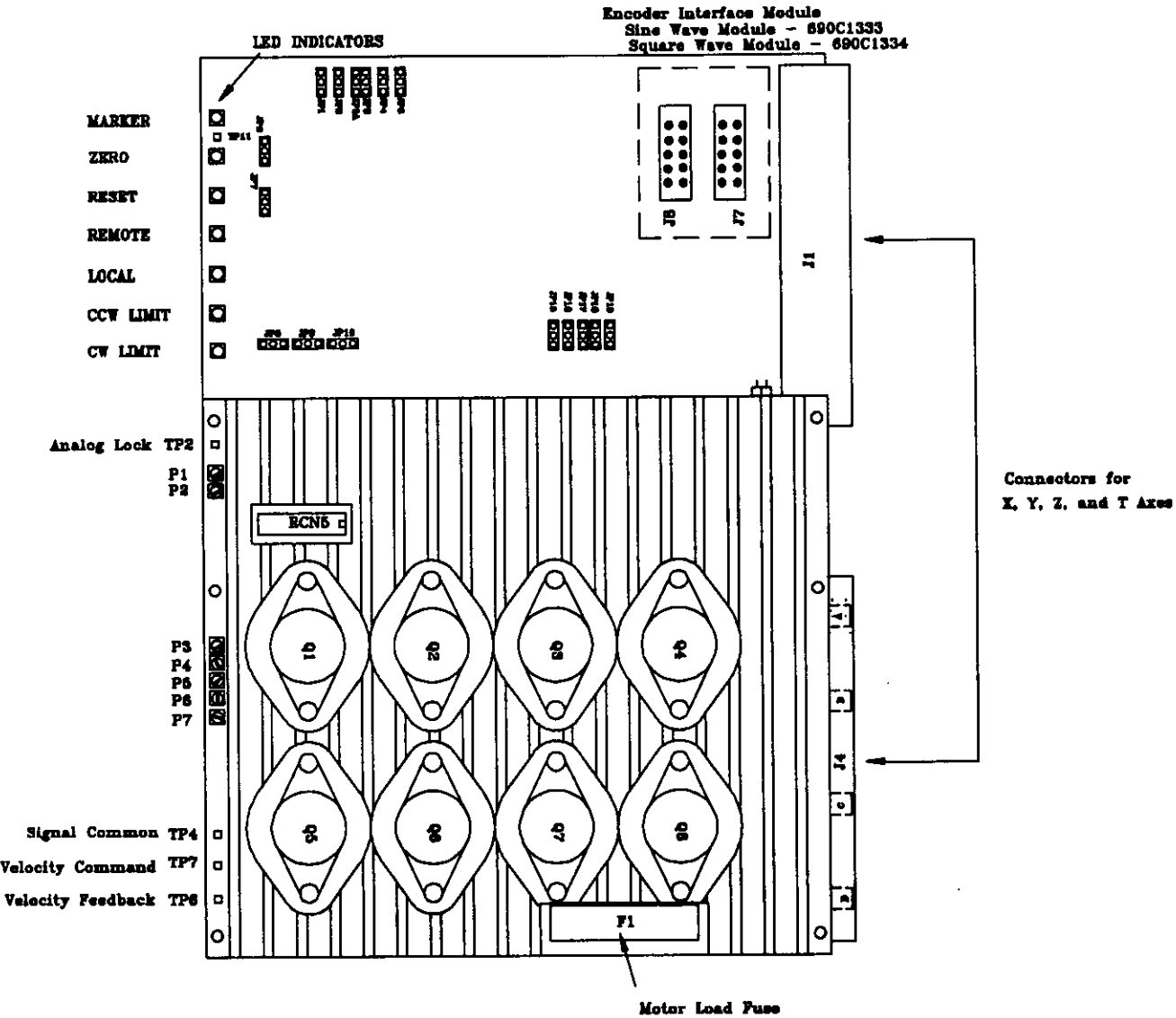


Figure 6-9: Outline of the DSL 3015 DC Servo Drive Module

DC DRIVE MODEL NO.	— DSL3015 —		— DSL4020 —		— DSL8020 —			— DSL16020 —		
	1017LT	1035LT	1050LT	1075LT	1135LT	1210LT	1410-02 -1600-99	1560LT	1960LT	
DC MOTOR MODEL NO.										
Continuous Torque (Max.)	17 0.12	35 0.25	50 0.35	75 0.53	135 0.95	210 1.5	410 2.9	560 4.1	800 5.6	
Peak Torque (Max.)	62 0.44	170 1.20	185 1.31	280 2.0	490 3.45	700 4.94	890 6.3	1300 9.2	2000 14.1	
Maximum Speed (RPM)	5000	5000	4200	5000	3600	2500	1800	1800	2400	
Motor Rotor Inertia Oz-In-Sec. ² Kg-M ²	0.004	0.0054 0.039×10^{-3}	.008 0.057×10^{-3}	.023 0.16×10^{-3}	.052 0.36×10^{-3}	.18 1.3×10^{-3}	.37 2.6×10^{-3}	.26 1.6×10^{-3}	.37 2.6×10^{-3}	
DC Drive Amplifier Settings										
DC Bus Voltage	20	40	40	80	80	80	80	160	160	
Max. Peak Current	15	20	20	20	20	20	20	20	20	
Output Fusing, Amps	4	4	5	5	5	6.25	6.25	6	10	
Maximum Input DC Drive Power (Watts)	80	108	145	170	240	260	395	713	1340	
Maximum Output Motor Shaft Power (Watts)	37	86	155	185	239	220	325	600	1190	
Weights										
Motor (Lbs.)	2.5	3.3	3.8	5.8	10.3	10.8	14.3	25.3	38	
Motor (Kg)	1.1	1.5	1.7	2.6	4.7	4.9	6.5	11.5	17.3	
Chassis										
U12S					50					
U12R					23					
U12H					42					
U12H					19					
					67 (80 with 2P12 dual supply)					
					26 (36 with 2P12 dual supply)					

* Must be equipped with additional cooling fan.

Table 6-3: DSL 3015, DSL 4020, DSL 8020 and DSL 16020 Specifications

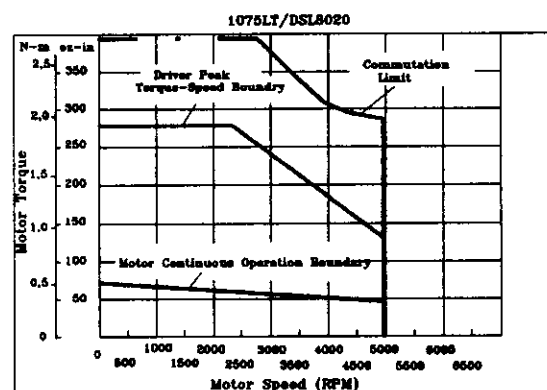
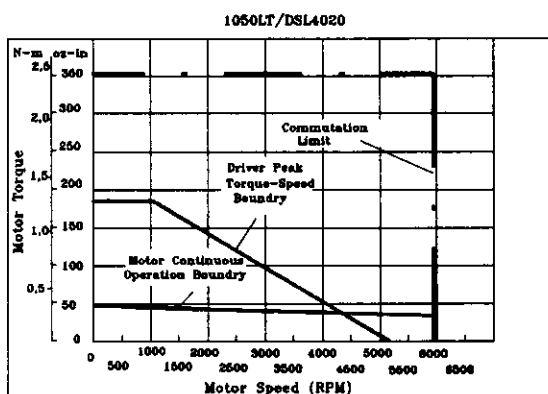
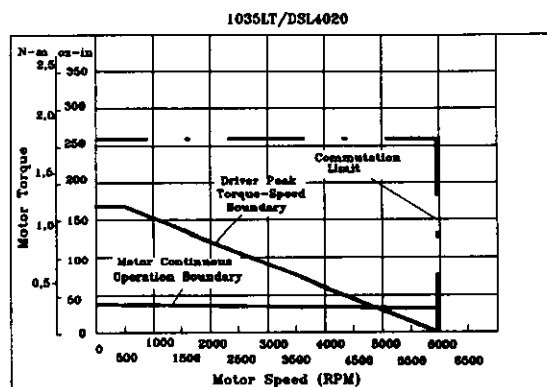
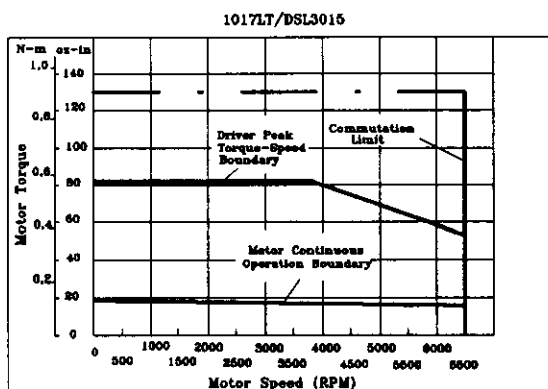


Figure 6-10: DSL Torque/Speed Curves

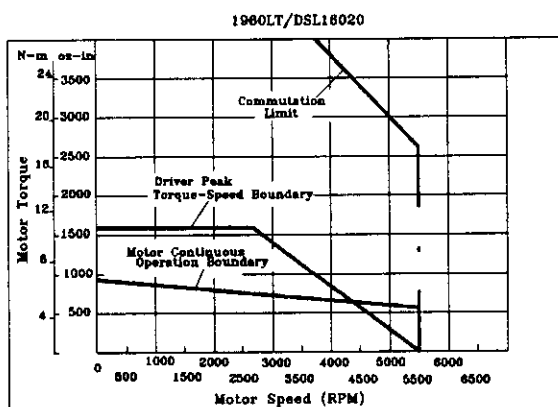
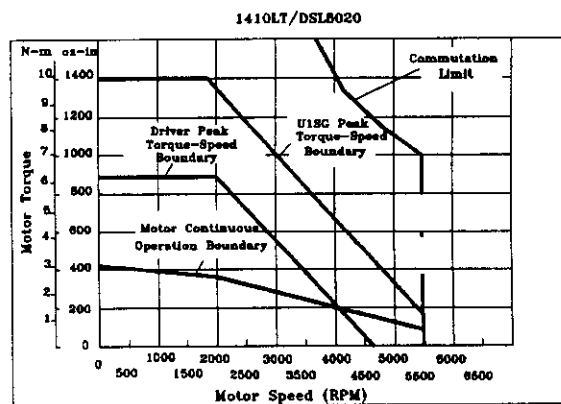
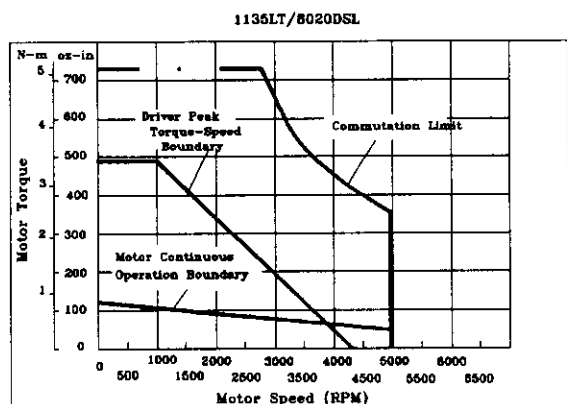


Figure 6-10 (con't): DSL Torque/Speed Curves

SECTION 6-3: DAV4008 & 16008 (AERODRIVE) STEPPING DRIVE MODULES

6-3-1: CIRCUIT DESCRIPTION

The DAV4008 and DAV16008 Drive Modules have been designed to accurately control a standard 1.8 degree Stepping motor with torque ranges of between 50 and 1200 oz-in under closed loop control. This module runs under closed loop control through an amplified sine wave feedback Encoder. Position and velocity loops are both "closed" with circuitry contained on this module. Analog "position locking" is obtained with the use of an amplified sine wave Encoder. Protection is included on this module for detecting faulty (or missing) Encoder feedback signals or excessive position command to position feedback error. Emergency shutdown is automatic under these conditions.

Analog position locking provides a very high level of Encoder feedback gain when the motor is within ± 1 encoder step of final position. With proper adjustment of the analog locking circuit, full motor torque may be generated with as little as .05 degree deflection of the motor shaft from the zero (or rest) position.

An outline of the DAV Drive Module is shown in Figure 6-14. Table 6-4 provides DAV Specifications and Figure 6-15 provides Torque/Speed curves.

The DAV Drive Module can be used in Unidex 14 chassis U14S, U14R and U14H.

6-3-2: DAV DRIVE MODULE OPERATION

The Aerodrive position control loop circuit works on the principle of command and feedback pulse summation. That is, CW and CCW command pulses are received from the control board module (described in the next section) and "summed" with the CW and CCW feedback pulses derived from the Encoder. The summation technique involves decoding logic, a counter to sum the feedback and command pulses and a D/A converter (Digital to Analog) whose output represents the position error between the summation of the command and feedback pulses. The position error signal is used as a velocity command signal to the pre-amplifier (described later). In addition to the position control circuit, (described above), an additional circuit is provided to "measure" the rate of CW and CCW feedback pulses per unit time in order to derive a representation for motor speed. The output of this circuit can be said to be a "electronically derived" velocity feedback signal from the motor.

For "locking" the motor at rest (zero count position error), a third circuit is provided to "hold" the DC motor firmly within a $\pm 1/2$ step of "rest". This function is termed Analog Lock, and applies to sine wave type Encoders only. The Analog Lock feature eliminates "one bit" jitter when the motor is in the rest position effectively canceling any external torque applied to the shaft, while maintaining "zero error" position.

A fourth circuit is provided on this board to "sum" the output signals of the three circuits just described. This circuit, labeled the "pre-amplifier" circuit, provides a signal as its output that is representative of the motor torque required to satisfy the position loop. This signal is actually the current command signal to the power amplifier portion of the Aerodrive (current drive directly proportional to torque). A block diagram of the four circuits described above is shown on Figure 6-11.

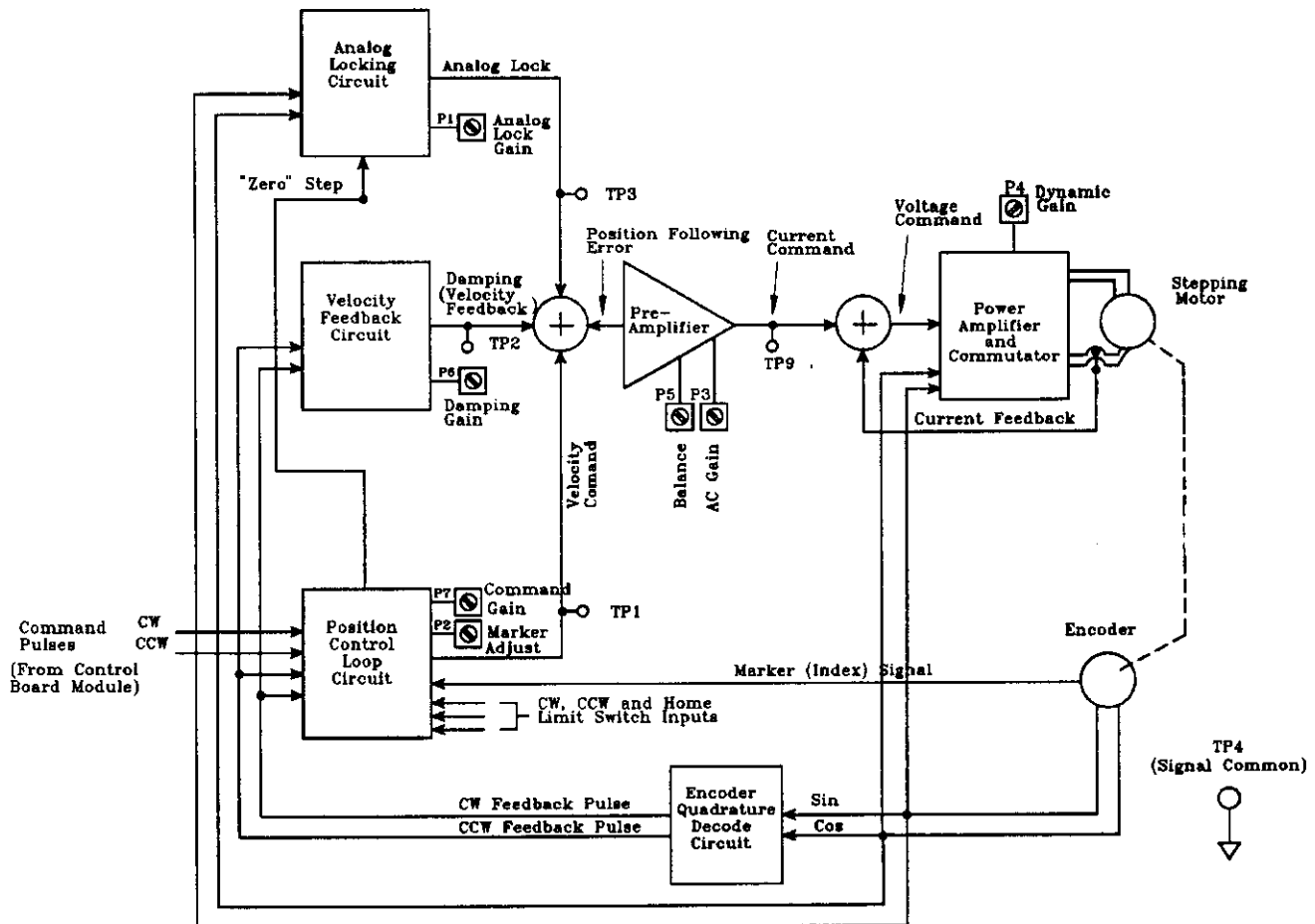


Figure 6-11: Block Diagram of DAV4008 and 16008

6-3-2-1: "HOME" REFERENCE DEFINITION & OPTIONS

All Unidex 14 Controllers are capable of generating a cold start reference position, which is the "Home" position. The basic "Home" cycle involves the following series of events. When the "Go Home" command is issued, the motor will turn CCW (standard) or CW (optional) until a "Home Limit Switch" activation occurs.

NOTE: The CCW and CW rotational references are viewed "looking" into the motor mounting flange.

Upon "Home Limit Switch" activation, the motor reverses and rotates in the opposite direction until the switch deactivates. If a "Marker" pulse option exists (see Section 6-3-2-2), the motor reverses and continues to rotate until the "Marker" pulse is located, and then stops.

The Home speed is determined by the value of the RCN5 6-11 resistor and will vary, depending on the system resolution.

Rotary Encoder (Steps/Rev)	Linear Encoder (Steps/mm)	Frequency	RCN5 6-11
200	50	2.5kHz	20K
400	100	5.0kHz	10K
1000 & up	250 & up	10.0kHz	5.1K

For most rotary motion stages, the "Home Limit Switch" referenced above is an independent switch incorporated specifically for the "Home" cycle. For linear motion stages, the "Home Limit Switch" is also often an independent switch. In most cases, the "CCW" or "CW" limit switches perform double duty and act as the "Home Limit Switch". The "Home Limit Switch" wiring to either the CW or CCW limit switch is done at the motor/stage, external to the Unidex 14.

It is not possible to use the "Home Limit" input at J13, 14, 19 or 20 when Opto 22 coupled limits are being used. When used in this fashion, "CCW" or "CW" limit input may be paralleled to the "Home Limit" input by reconfiguring jumper JP5 on the Aerodrive PC board. The jumper definition is as follows:

JP3 HOME LIMIT SOURCE:

(Standard) From Home Limit Input: Jumper JP5-3 to 4 , Remove JP5- 1 to 2, JP5-5 to 6.

(Optional) From CCW Limit Input: Jumper JP5-1 to 2, Remove JP5-3 to 4, JP5-5 to 6.

(Optional) From CW Limit Input: Jumper JP5-5 to 6, Remove JP5-1 to 2, JP5-3 to 4.

See Figure 6-14 for jumper locations.

When the Home command is issued the standard direction of motor rotation is CCW. If CW rotation is required, jumper JP3 on the Aerodrive board must be reconfigured.

JP1 HOME DIRECTION JUMPER:

(Standard) CCW Home - Jumper JP3-1 to 2, Remove JP3-2 to 3.

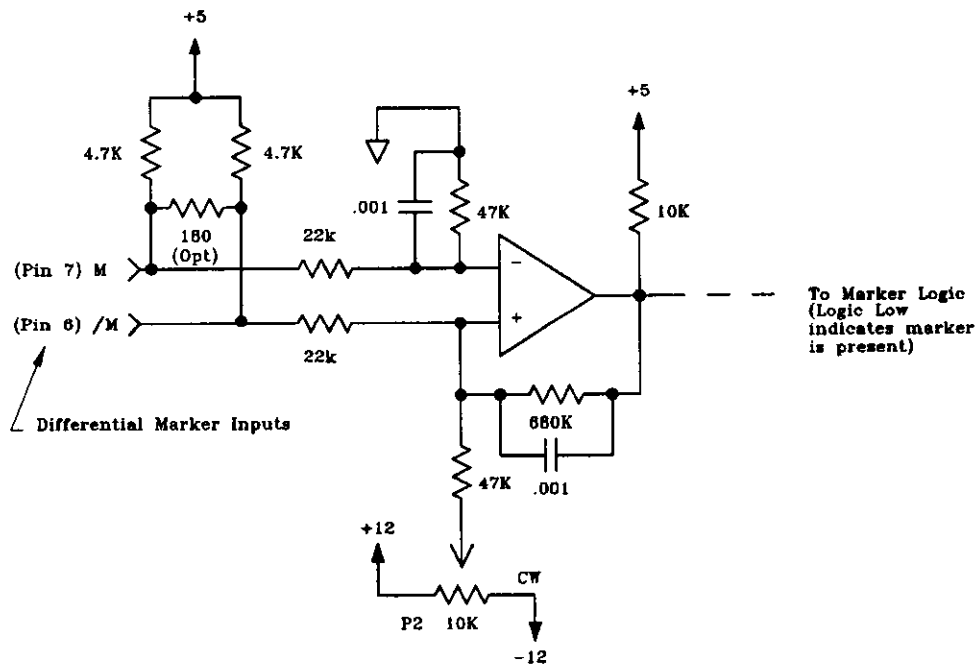
(Optional) CW Home - Jumper JP3-2-3, Remove JP3-1 to 2.

See Figure 6-14 for jumper locations.

NOTE: Positive move commands to the Unidex 14 causes the motor to rotate in the CCW direction.

6-3-2-2: MARKER INPUT CIRCUIT

Shown below is a circuit diagram of the differential Input Marker circuit:



Potentiometer P2 in the circuit diagram above allows the threshold of the Input Marker signal to be adjusted. The threshold level is increased by turning P2 CCW. To activate the Marker logic, the voltage signal at "M" must rise slightly higher than the voltage signal at "/M", plus the threshold setting at P2. In other words:

$$M /M + P2 \text{ (threshold)}$$

To adjust the marker, it is recommended that the Aerodrive is given a Home command. This will maintain a motor "search" for the CW, CCW or Home Limit Switch. When the limit is reached, the motor will reverse directions and the Aerodrive will seek the first marker pulse. Pot 2 should then be adjusted such that the pulse is successfully detected by the Aerodrive homing logic.

6-3-2-3: LIMIT SWITCH POLARITY SELECTION

As a standard, Unidex 14 controllers are configured to interface to normally open (active low) limit switches (CCW, CW and Home). If use of normally closed (active high) limit switches is required, jumpers JP2, JP4, and JP5 on the Aerodrive board must be reconfigured. The jumper definitions follow:

LIMIT/LIMIT-N JUMPERS (JP2, JP4, JP5)

- (Standard) CCW Limit, Normally Open - Jumper JP7-2 to 3, Remove JP7-1 to 2.
- (Optional) CCW Limit, Normally Closed - Jumper JP7-1 to 2, Remove JP7-2 to 3.
- (Standard) CW Limit, Normally Open - Jumper JP4-2 to 3, Remove JP4-1 to 2.
- (Optional) CW Limit, Normally Closed - Jumper JP4-1 to 2, Remove JP4-2 to 3.
- (Standard) Home Limit, Normally Open - Jumper JP6-2 to 3, Remove JP6-1 to 2.
- (Optional) Home Limit, Normally Closed - Jumper JP6-1 to 2, Remove JP6-2 to 3.

See Figure 6-14 for jumper locations.

6-3-2-4: ENCODER MULTIPLICATION PARAMETERS

Incremental Encoders produce two output signals generally referred to as sine and cosine. These signals are displaced 90 degrees with respect to each other (i.e., in quadrature with respect to each other). These quadrature signals are in the form of amplified sine waves.

Through electronic decoding logic on the Aerodrive, these quadrature signals can be interpreted as providing 1, 2 or 4 individual steps (machine steps) per one full cycle of sine and cosine.

Multiplying the Encoder signal by 1, 2, or 4 (X1,X2,X4), involves the manipulation of jumpers and, in case of amplified sine wave Encoders, the changing of resistor values on the Encoder interface module (see Figure 6-14).

ENCODER MULTIPLICATION	MULTIPLIER	R5	R6	R7	R8	JUMPER	JUMPER JP9	JUMPER JP8	JUMPER JP15	JUMPER JP14
SINE WAVE 690C1333	X1	22K	OUT	OUT	43K	2-3	2-3	2-3	2-3	2-3
	X2	22K	OUT	OUT	43K	2-3	2-3	1-2	2-3	2-3
	X4	OUT	43K	22K	OUT	1-2	1-2	2-3	2-3	NONE
Encoder Interface Module							Main Board			

NOTE: Changing resolution will also change system gain. For optimum performance, it may be necessary to change the Accel./Decel. parameter and/or the gain setting of the DAV Drive Module.

6-3-2-5: ADJUSTMENTS

Seven potentiometers and six test points are provided on the Aerodrive module. A description of these pots and test points are as follows (refer to Figure 6-14).

NOTE: If the Aerodrive module is supplied with an Aerotech motor, all adjustments have been factory set for that motor, further adjustments should not be necessary.

P1 ANALOG LOCK GAIN ADJUST

This pot adjusts the sine lock gain for analog lock in the "zero" step (rest) position. Analog lock is adjusted by monitoring testpoint TP3 (with respect to signal common, TP4) and the zero LED indicator. Turning P1 CCW increases the analog lock gain.

When the motor is at rest (no command pulses), the "ZERO" LED indicator must be energized by adjusting the BALANCE pot P5 (discussed below). When the ZERO LED is energized, the analog lock testpoint (TP3) should be monitored (with an oscilloscope) and both the BALANCE pot P5 and the ANALOG LOCK pot P1 should be adjusted until TP3 reads close to zero volts.

P2 MARKER ADJUST

This pot adjusts the threshold of the Marker (index) input pulse from the incremental Encoder.

P3 AC GAIN ADJUST

This pot adjusts the AC gain of the pre-amplifier. Pots P7 and P6 affect the DC as well as the AC gain of the pre-amplifier as discussed below. Turning the AC GAIN pot (P3) CCW increases the AC gain of the pre-amplifier.

P4 DYNAMIC (DYN) GAIN

This pot adjusts the motor commutation angle relative to motor speed for the purpose of increasing motor torque at higher speeds. Turn the pot CW to increase the commutation angle relative to motor speed.

The motor current feedback signal, which is summed with the current command signal to produce a voltage command signal to the power amplifier (see Figure 6-14), can be monitored at testpoint TP9.

Both the Current Command and Current Feedback testpoints (TP8 and TP9 respectively) provide signal gain ratio of ± 4.3 amps per volt.

The summation circuitry for Current Feedback and Current Command (specifically an integral-lead circuit) provides a motor current bandwidth of approximately 1000 Hz (assuming a 2 - 5 milliHenry load inductance).

P5 BALANCE POT

This pot adjusts analog offsets inherent in the pre-amplifier circuit (see Figure 6-13) which can cause position error. This pot is adjusted when the motor is at rest. Set the pot to a position such that the "zero" LED remains lit. Test the setting by making a move and confirming that the "zero" LED lights when the motor comes to a stop.

P6 VELOCITY FEEDBACK GAIN

This pot adjusts the amount of DC Velocity Feedback Gain to the pre-amplifier. Velocity Feedback is electronically derived from the Encoder (see Figure 6-14). Velocity Feedback has two purposes. First, to provide "damping" to the position control loop to eliminate position loop oscillation. Secondly, the Velocity Feedback provides a means of generating position "following error" in the position loop. Following error provides the desirable effect of eliminating position "overshoot" when operating the motor at high Accel./Decel. rates. Testpoint TP6 provides a monitor for the Velocity Feedback signal. Turning this pot CW increases Velocity Feedback Gain and typically reduces motor positioning response.

P7 COMMAND GAIN

This pot adjusts the amount of DC Velocity Command Gain to the pre-amplifier. Command Velocity is derived from the accumulated difference (or error) between the command pulses from the control board module and the feedback pulses generated by the Encoder. Testpoint TP1 provides a monitor for the Velocity Command signal. The ratio of the voltage of this signal with respect to actual command motor velocity varies with the encoder resolution and the setting of the Velocity Feedback Gain described above, and therefore cannot be directly defined.

NOTE: Adjustment of the COMMAND VELOCITY GAIN pot (P7) and FEEDBACK VELOCITY GAIN pot (P6) may require that the BALANCE pot (P5) be readjusted to obtain the "ZERO" step condition discussed earlier.

6-3-2-6: ADJUSTING POSITION & VELOCITY LOOP

The DAV4008 and DAV16008 Drive Modules are factory adjusted for given Aerotech Aerodrive motor(s) (i.e., 50SM, 310SM, etc.) shipped with the Unidex 14. *If a user-supplied motor and Encoder is to be used with Unidex 14, Aerotech should be notified of the motor and Encoder parameters at the time of purchase to insure compatibility and proper factory set up.*

If the Motor, Encoder and Translation stage are user supplied, the following adjustment procedure is recommended:

DANGER: PRIOR TO MAKING ANY ELECTRICAL CONNECTIONS OR DISCONNECTIONS. MAKE CERTAIN ALL ELECTRICAL POWER SWITCHES ARE IN THE "OFF" POSITION.

NOTE: An oscilloscope is required for the following procedure.

1. If the Motor, Encoder and optional Tachometer are one assembly, disconnect them from the load, if possible.
2. Turn P1 full CW, P3 full CW, P4 mid-range, P5 midrange, P6 full CW, and P7 full CCW. This will set the gain to the lowest adjustable value. Apply power.
3. Write a short program that will run Unidex 14 back and forth with a 1-second dwell between moves. Set the feedrate for 1/3 of the maximum system speed (typically for Aerotech supplied Motors command a velocity of 40,000 steps/second). Set the Accel/Decel rate $400,000 \text{ steps/sec}^2$, linear profiling. Make certain that the move is long enough to affirm that the Motor can accelerate and run at programmed speed.

4. Adjust P7 (COMMAND GAIN ADJUSTMENT) CW for less than or equal to $\pm 4V$ of velocity command at testpoint TP1 with respect to testpoint TP4 (SIGNAL COMMON). If the motion stops, the range of the D/A has been exceeded. If this is the case, turn P7 full CW, turn the power OFF and then ON, clearing the fault condition and repeat the preceding procedure. If the range of P7 does not prove sufficient, turn P6 (DAMPING) CCW to decrease following error.

If the fault condition persists, the problem may be due to one of the following:

- a) The requested program speed exceeds the capabilities of the Motor and Amplifier.
 - b) There may not be enough acceleration torque available to accelerate the Motor and/or load up to speed. If this is the case, it may be necessary to decrease the Accel/Decel rate from by a factor of two.
 - c) The velocity command is too high with respect to the Dynamic gain pot setting. Turn the Dynamic gain pot further CW.
5. If the load was disconnected in Step 1, reconnect it at this time.
 6. Turn the AC gain adjustment, P3, slowly CCW, such that the system begins to oscillate. Turn P3 slowly CW such that system oscillation stops then continue to turn P3 CW 1/8 turn more.
 7. The system should now be stable and the gain adjustments very conservatively set (OVER DAMPED). At this point the system should be running the program of Step 5 very smoothly. To optimize positioning time, run the program of Step 5 at maximum system speed (typically 90,000 steps/second) and turn P7 (COMMAND GAIN) further CW, to achieve the results at testpoint TP1 (VELOCITY COMMAND) as shown in Figure 6-12. If the range of P7 is not sufficient, it may be necessary to turn P6 (DAMPING) further CCW. If adjustment is made to P6 you must also readjust P3 (AC GAIN). (Follow the procedure outlined in Step 6.)

8. Increase the Accel/Decel rate until you start to notice overshoot at testpoint TP1 (VELOCITY COMMAND).
9. To adjust the Balance it is necessary to adjust the P1 (ANALOG LOCK GAIN) and P5 (BALANCE ADJUSTMENT). With the system at rest, adjust P5 until the "Zero" indicator LED is lit on the front of the Unidex 14. Connect the probe of the oscilloscope to testpoint TP3 (LOCK SIGNAL) and the common of the oscilloscope to testpoint TP4 (SIGNAL COMMON). Turn P1 (ANALOG LOCK GAIN) mid-range and adjust P5 (BALANCE) for zero volts ($\pm 0.5V$) at testpoint TP3. Execute a one step move and adjust P1 for approximately 25% overshoot at testpoint TP3 upon completion of the move with little or no ringing (as shown in Figure 6-13). After adjusting P1 it may also be necessary to readjust P5.

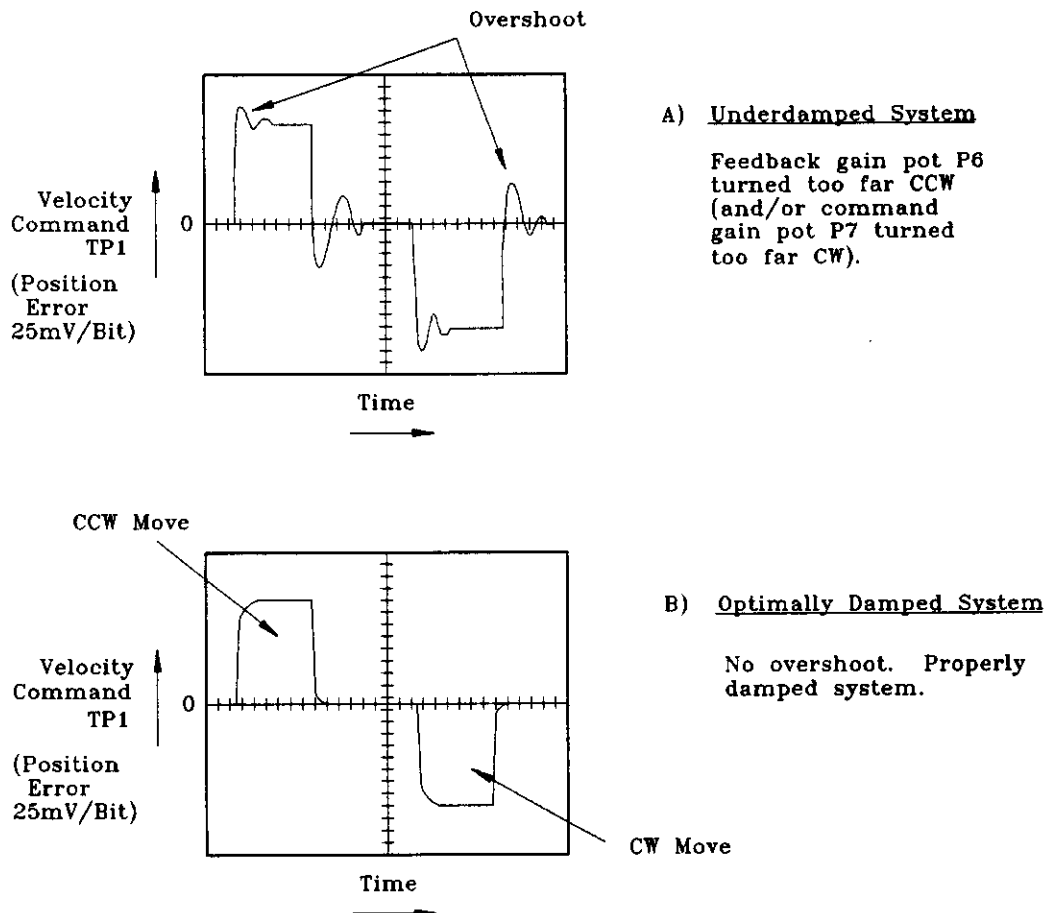


Figure 6-12: Velocity Command refers to TP1 for Underdamped (A), and Optimally Damped (B), Position Loop Response

NOTE: The velocity command shown in Figure 6-12 also represents the position error which is scaled to a 25mV/machine step.

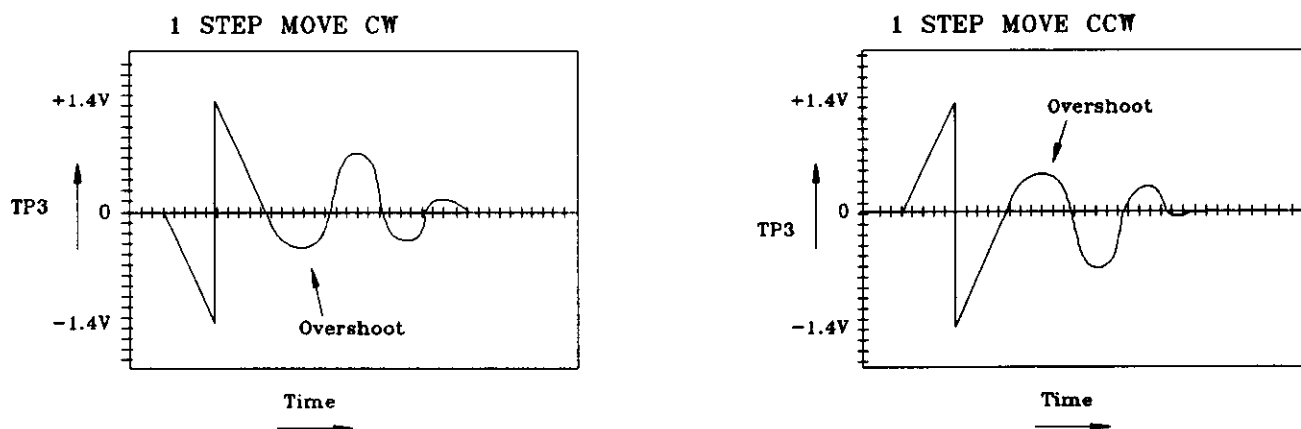


Figure 6-13: Analog Lock Signal at TP3

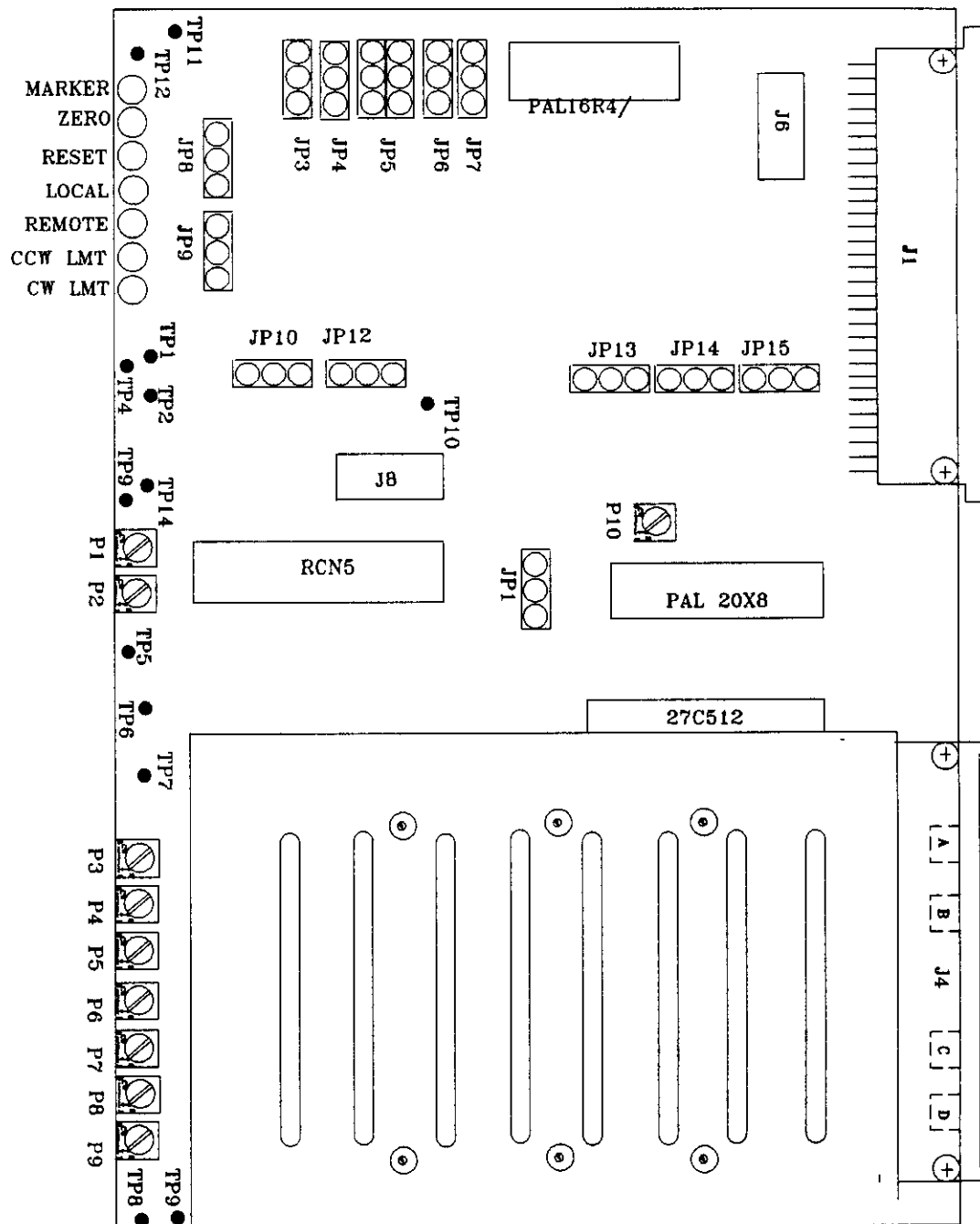


Figure 6-14: Outline of the DAV4008 and DAV16008 Drive Module

AERODRIVE P/N	DAV4008	DAV4008	DAV16008	DAV16008
MOTOR P/N	50SM	101SM	310SM	1010SM
STATIC TORQUE (PEAK)	55	200	570	1150
OZ.IN. (CONTINUOUS)	38	90	370	1050
PEAK TORQUE AT 2000 RPM (OZ.IN.)	10	55	150	200
MOTOR OUTPUT POWER (WATTS)	14	62	290	400
MOTOR ROTOR INERTIA				
OZ-IN-SEC ²	1.66E ⁻³	5E ⁻³	26.5E ⁻³	114E ⁻³
KG-M ²	11.8E ⁻⁶	35E ⁻⁶	187E ⁻⁶	805E ⁻⁶
MOTOR FRAME NEMA 2	23	23	34	42
POSITION RESOLUTION	2000 STEP/REVOLUTION (STANDARD)			
DRIVE VOLTS (DC)	40	40	60	160
AMPS	1	5	6	8.8
TYPE	BIPOLAR	BIPOLAR	BIPOLAR	BIPOLAR
WEIGHTS				
MOTOR (LBS.)	2.3	3.6	8.1	20
(KG)	1.04	1.63	3.67	9.07
MOTOR/DRIVE (LBS.)	17.3	18.6	23.5	30
(KG)	7.85	8.44	10.66	13.61
(INCLUDES ALL INTER- CONNECTING CABLES)				
INPUT POWER				
VOLTS (50/60 HZ)	115/220-240	115/220-240	115 ONLY	115 ONLY
AMPS	.5	1	3	4
(ALLOWABLE VOLTAGE TOL- ERANCE- +/-10% MAX)				

Table 6-4: DAV4008 and DAV16008 Specifications

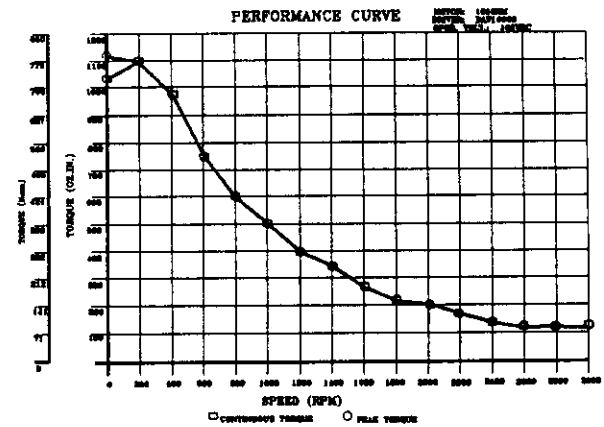
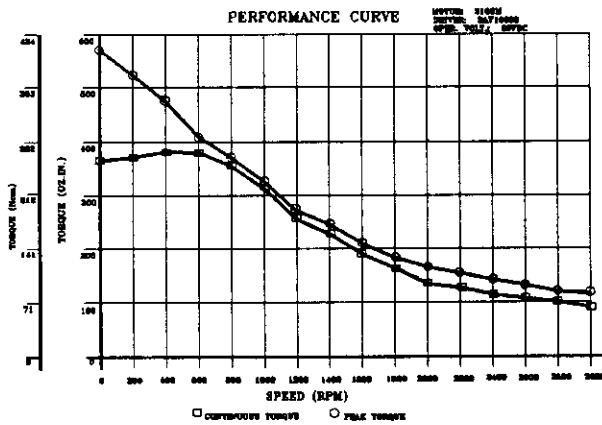
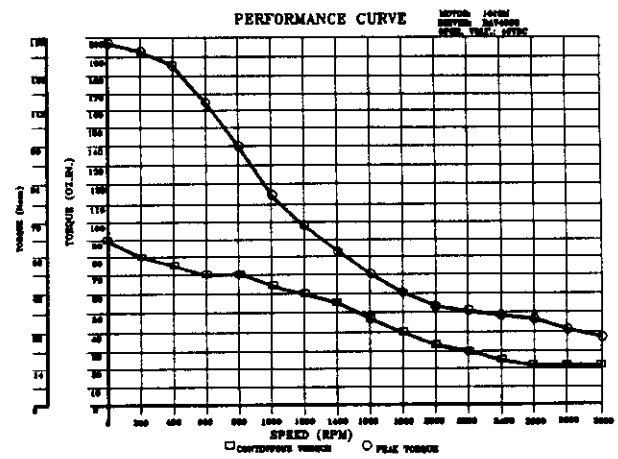
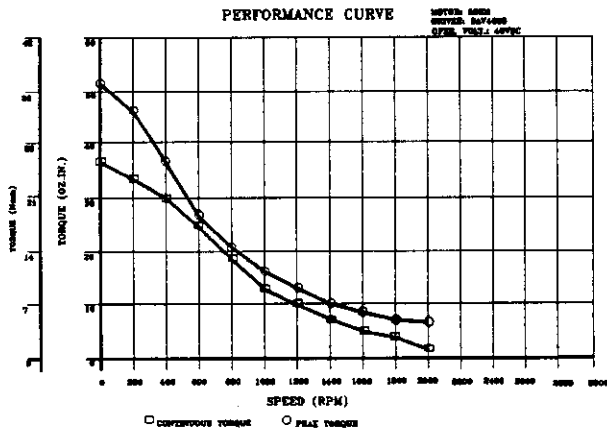


Figure 6-15: DAV4008 and DAV16008 Torque/Speed Curves

SECTION 6-4: POWER SUPPLY BOARD

6-4-1: CIRCUIT DESCRIPTION

The Power Supply board (module) is provided for use with the DMV8008, DMV16008, the DSL3015, DSL4020, DSL8020 and DSL16020, as well as the DAV4008 and DAV16008. This module provides +5VDC and ± 12 VDC control voltages to the DMV, DSL and DAV Drive modules (note that D3001, D1401, DM1501, DM4001, DM4005 and DM6006 Stepping Drive modules do not require this board for control power generation since all control and motor supply voltages are generated on the board).

An outline of the Power Supply board is shown in Figure 6-16.

A DC Shunt Regulator circuit is also provided on the Power Supply board to clamp excessive regenerative energies present when motors with high inertial loads are decelerating. The Shunt Regulator (slot P5) can only accommodate one DC bus power supply & must shunt the power supply associated with Drive module slot P4. If P3, P2 and P1 share the same supply as slot P4, these slots also share shunt regulation.

The Shunt Regulator circuit is factory set to operate with DC Bus voltages of ≤ 80 V or 160V (DMV16008, DSL16020 and DAV16008 only).

Referring to Figure 6-16, it can be noted that eight testpoints are provided as monitor points for the user. These testpoints are described as follows:

- TP5** +5VDC (for Encoders, Optical Markers, or Optical Limit Switches)
- TP6** +5VDC (for DMV and DSL Series Drive Modules)
- TP8** +12VDC (for DMV and DSL Series Drive modules)
- TP7** -12VDC (for DMV and DSL Series Drive modules)
- TP4** Signal common
- TP1** + DC bus voltage (Shunt Regulator bus voltage)
- TP3** + DC bus voltage return (reference point for Shunt Regulation)
- TP2** Shunt Regulation switching reference

The voltage at which the Shunt Regulator circuit begins to function (the Shunt Voltage Threshold) is adjustable via potentiometer P2 (see Figure 6-16). This pot is set at the factory for a threshold of 95 VDC for all bus voltages of 80V or less, and for a threshold of 185VDC for a bus voltage of 160VDC.

If it is necessary to change these settings, the Shunt Regulator Threshold can be readjusted by monitoring TP2 with respect to TP3. The voltage on TP2 will switch between the given bus voltage and zero when the Shunt Voltage Threshold is reached. Turning P2 CCW decreases the Voltage Threshold point, and turning P2 CW increases the Voltage Threshold point. The range of P2 adjustment is from 80VDC to 200VDC. The Shunt Voltage Threshold is typically adjusted by first accelerating and decelerating all motors simultaneously at full speed while monitoring TP2. The regenerative bus voltage level should not be allowed to exceed 20% of the rated drive voltage while the motors are decelerating. Note that excessive shunt regulation will cause F5 to open.

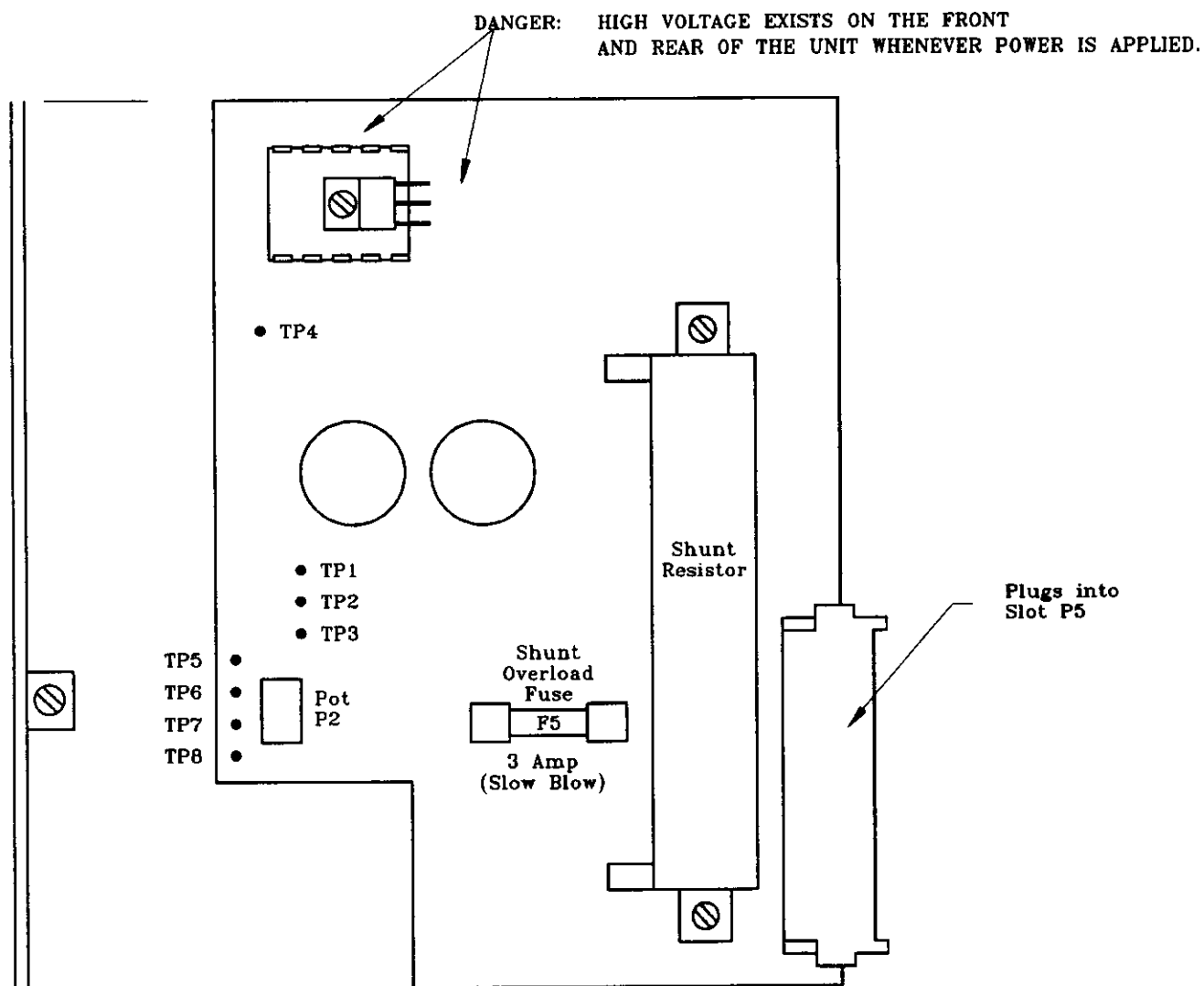


Figure 6-16: Outline of the Power Supply Board Module

CHAPTER 7: STEPPING DC SERVO & AERODRIVE MOTORS

SECTION 7-1: HARDWARE SPECIFICATIONS

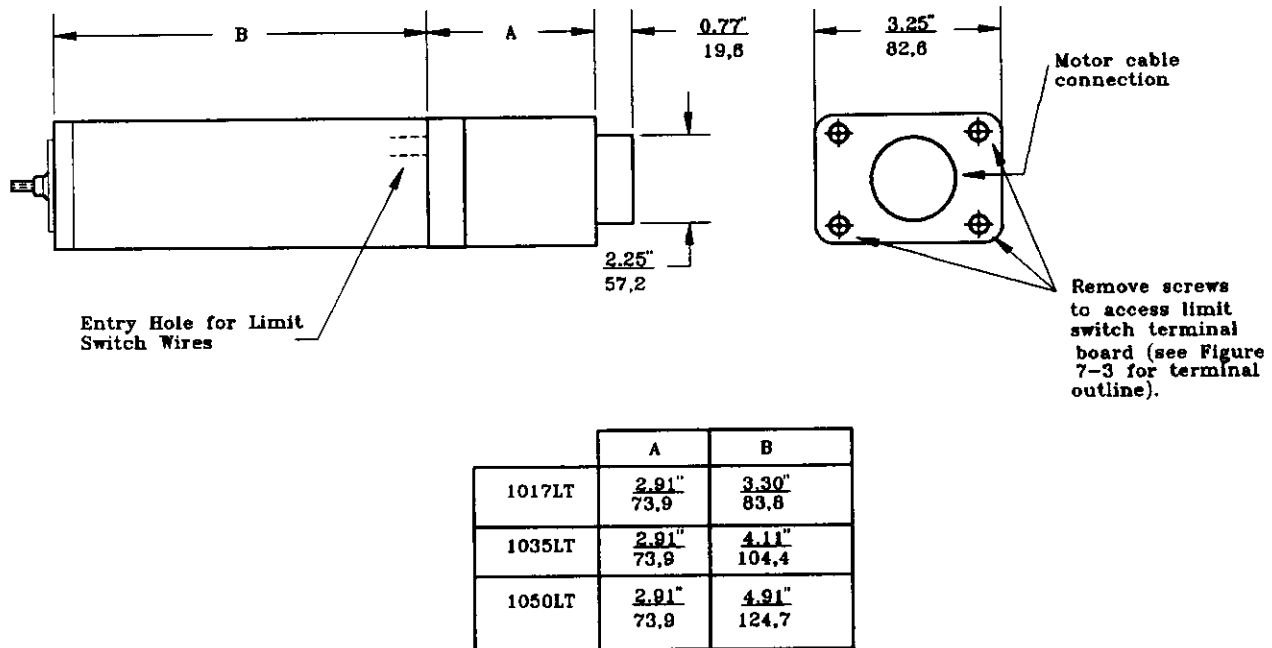
Dimensional data for Stepping, DC Servo and Aerodrive motors used with the Unidex 14 family of Drives is illustrated in this section.

Figure 7-1 shows dimensional data for the DC Servo motors. Figure 7-2 shows dimensional data for the Stepping motors. Detailed mechanical data such as shaft lengths and shaft keying are not shown in these figures. This data is supplied in separate documents.

Figure 7-3 shows pin-out assignments for the Limit Switch interface board associated with most of the motors illustrated in Figure 7-1 and 7-2.

Figure 7-4 provides dimensional data for the Unidex 14 Control enclosures.

MS01 - 1017LT, 1035LT, 1050LT Motors



MS01 - 1075LT, 1135LT Motors

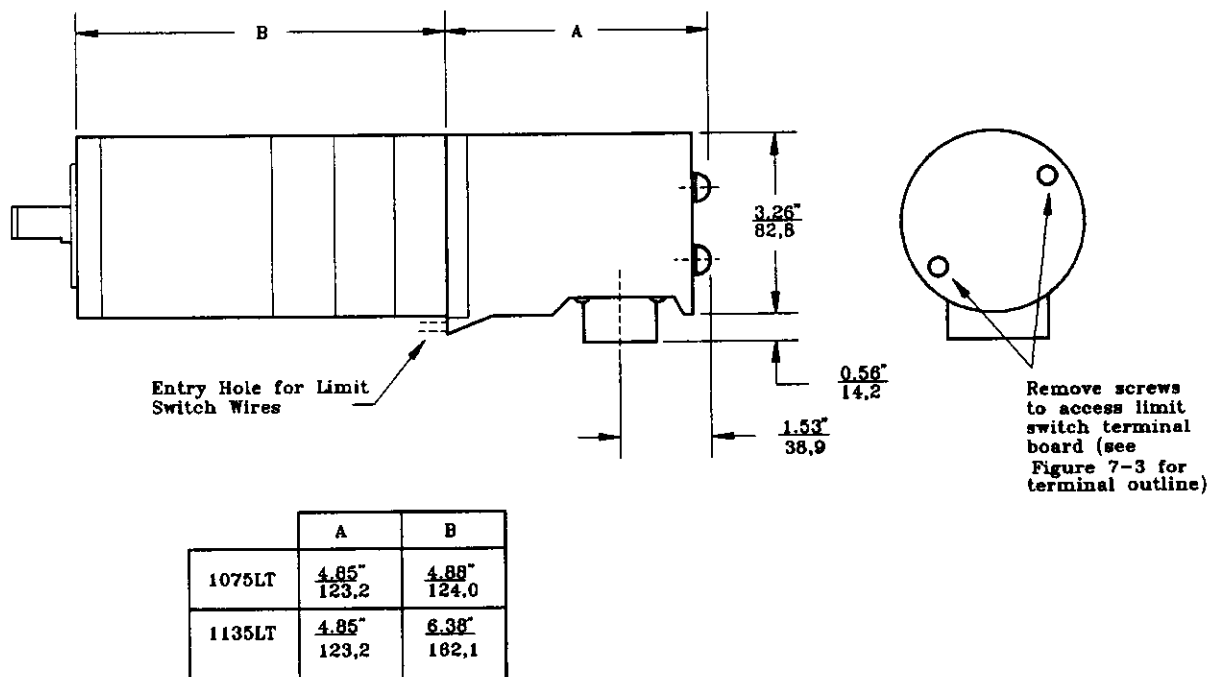


Figure 7-1: Mechanical Dimensions for Unidex 14 DC Motors

MS01 - 1210LT, 1410LT Motors

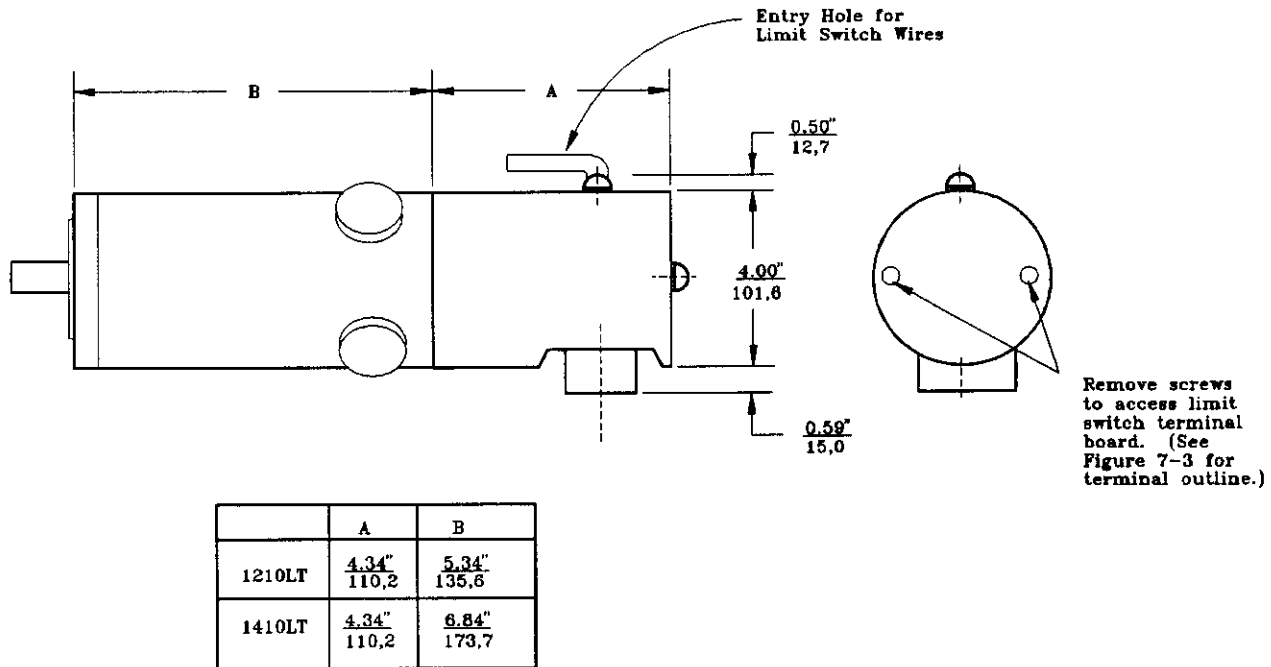
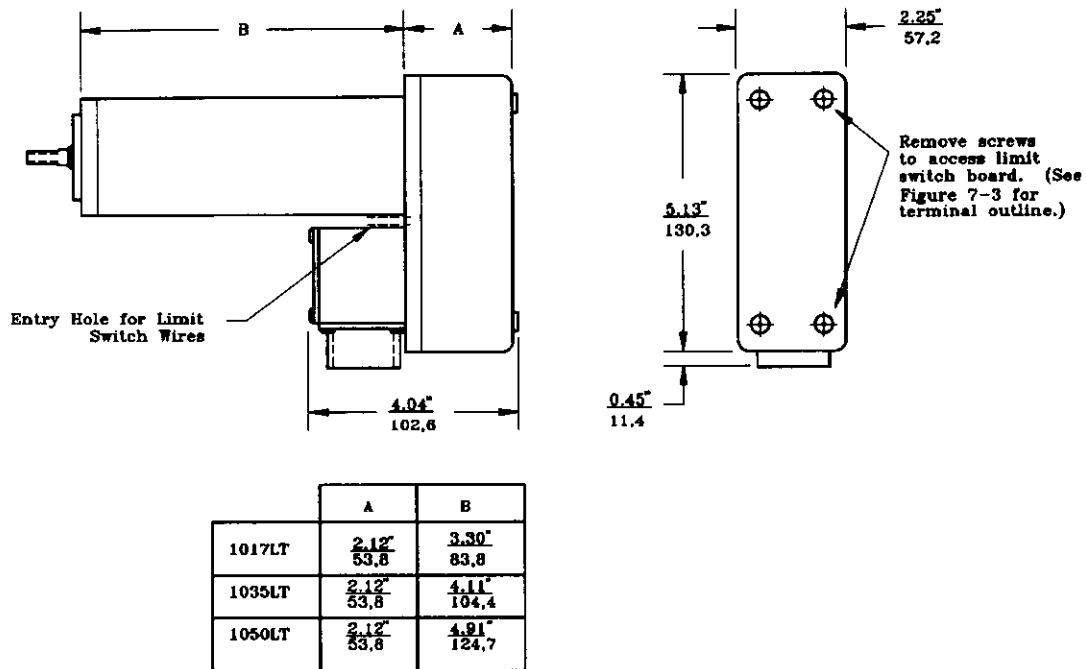


Figure 7-1: Continued

MSOF - 1017LT, 1035LT, 1050LT Motors



MSOF - 1075LT, 1135LT Motors

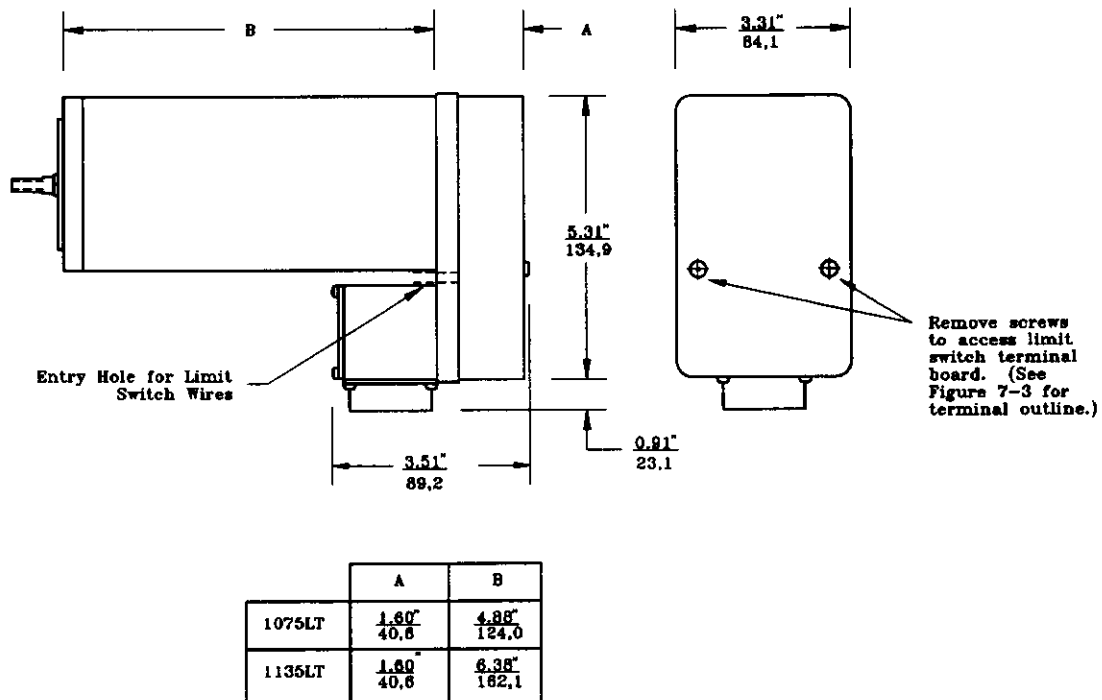
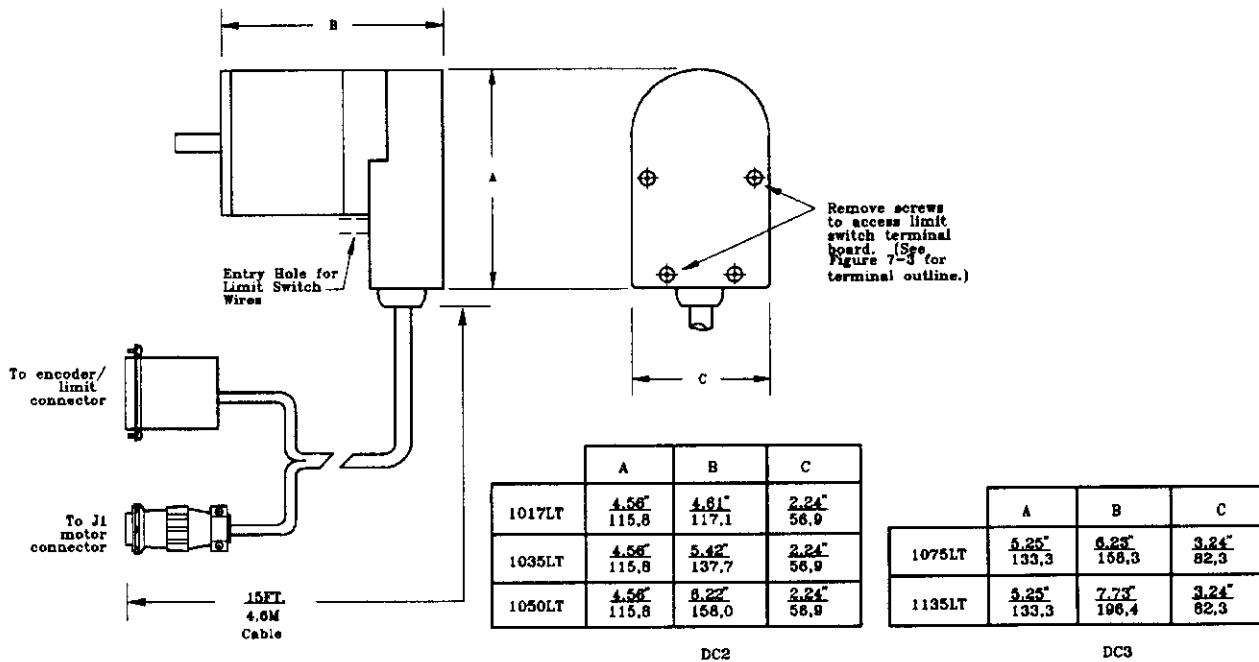


Figure 7-1: Continued

DC2 - 1017LT, 1035LT, 1050LT Motors
DC3 - 1075LT, 1135LT Motors



DC2E - 1017LT, 1035LT, 1050LT

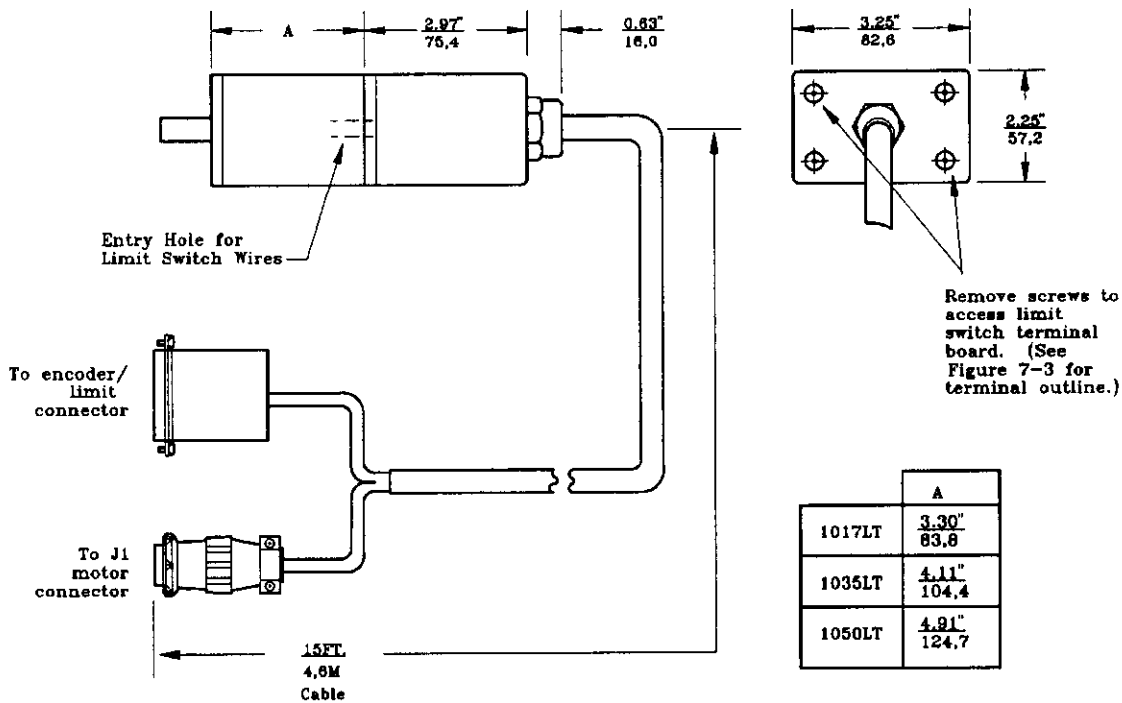


Figure 7-1: Continued

DC3E - 1075LT, 1135LT

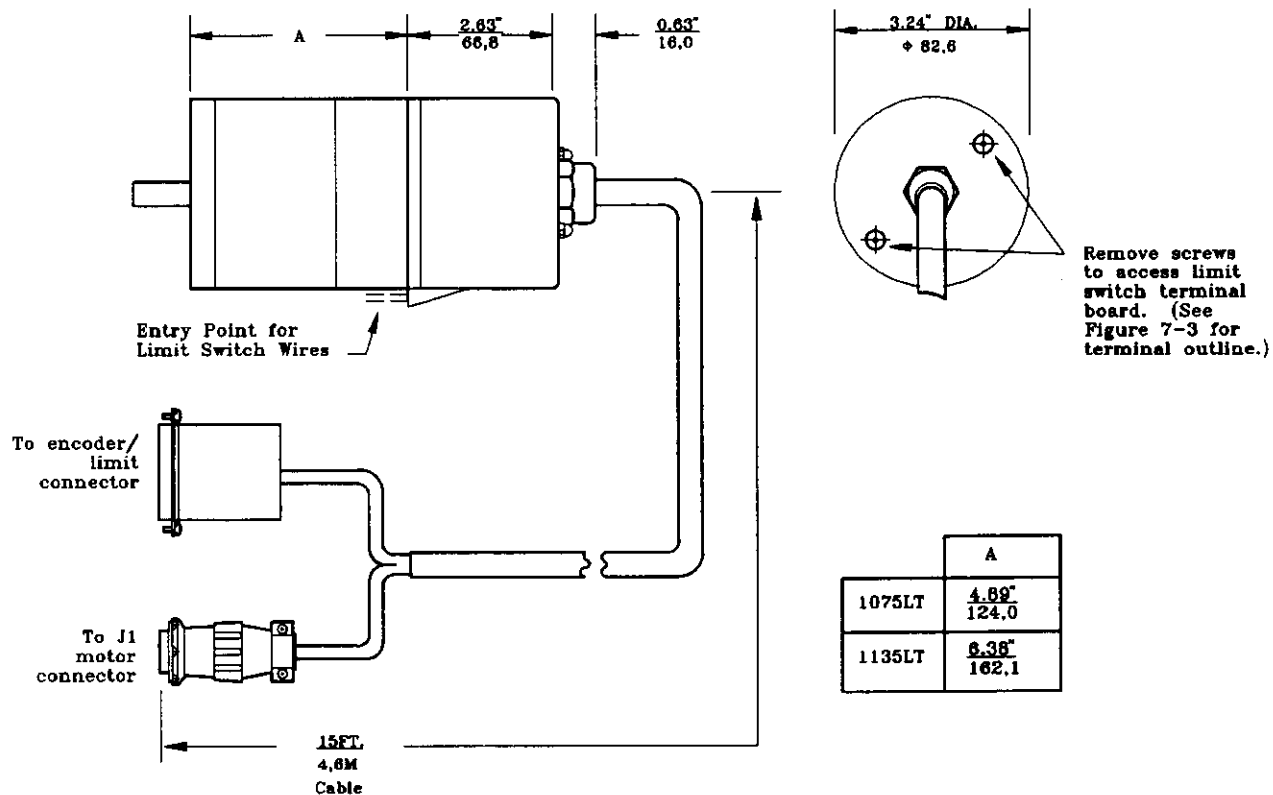
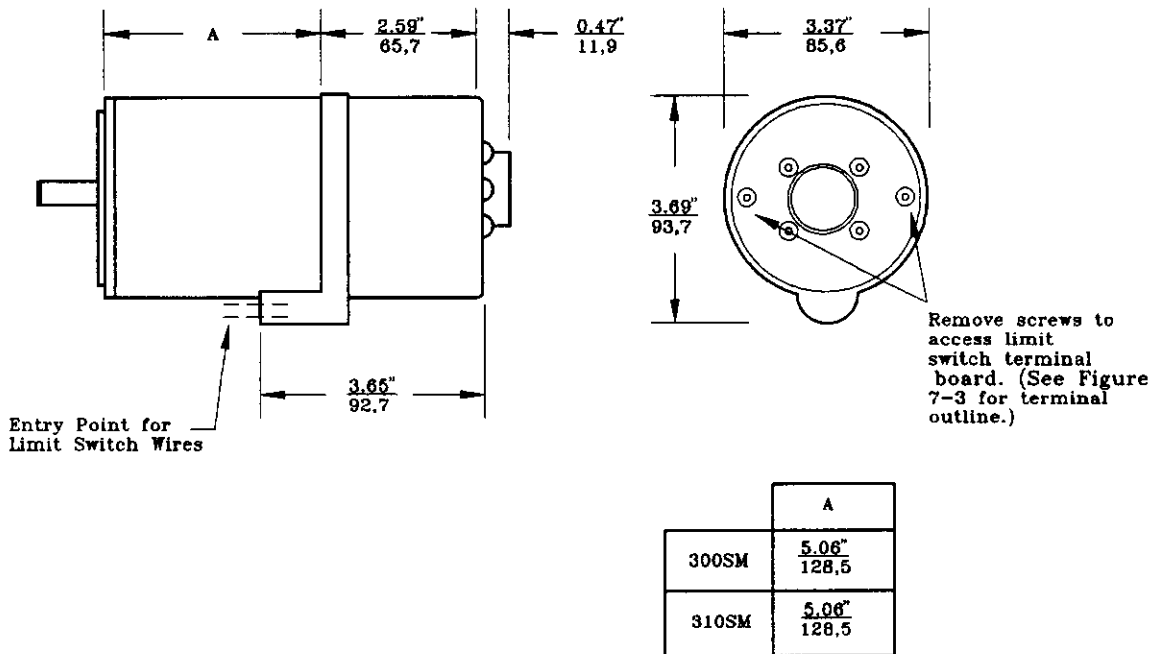


Figure 7-1: Continued

B3E-HM - 300SM, 310SM Motors
B3E-EXXXAS 310



B2E-HM - 50SM, 101SM Motors
B2E-EXXXAS 50 or 101

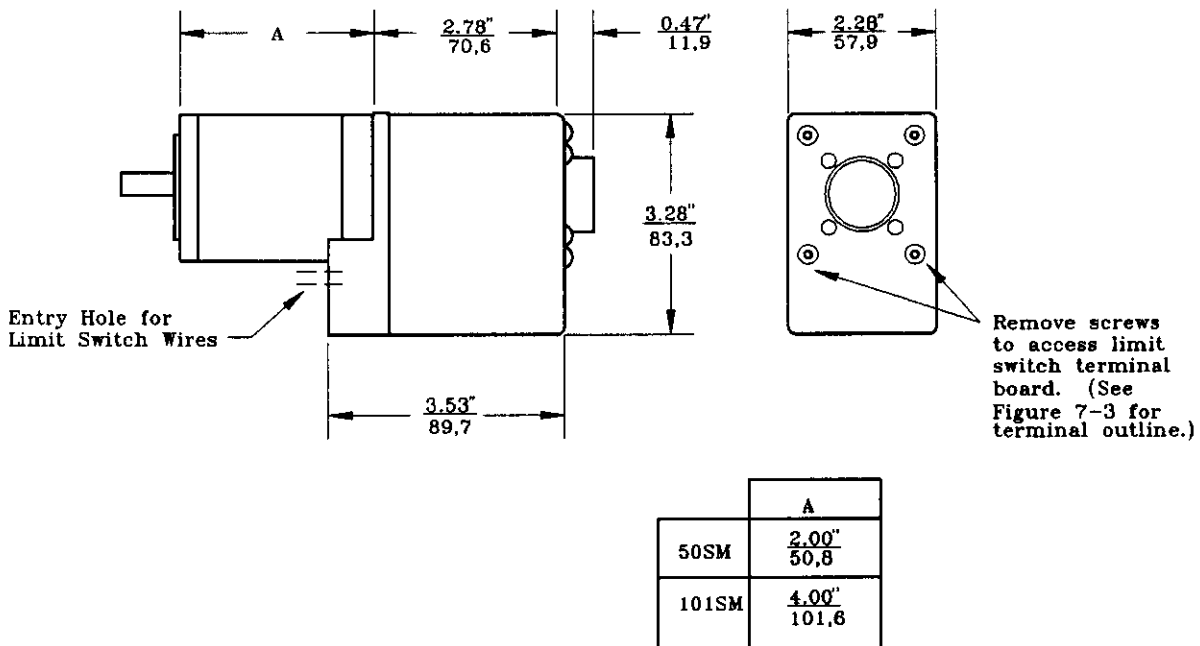
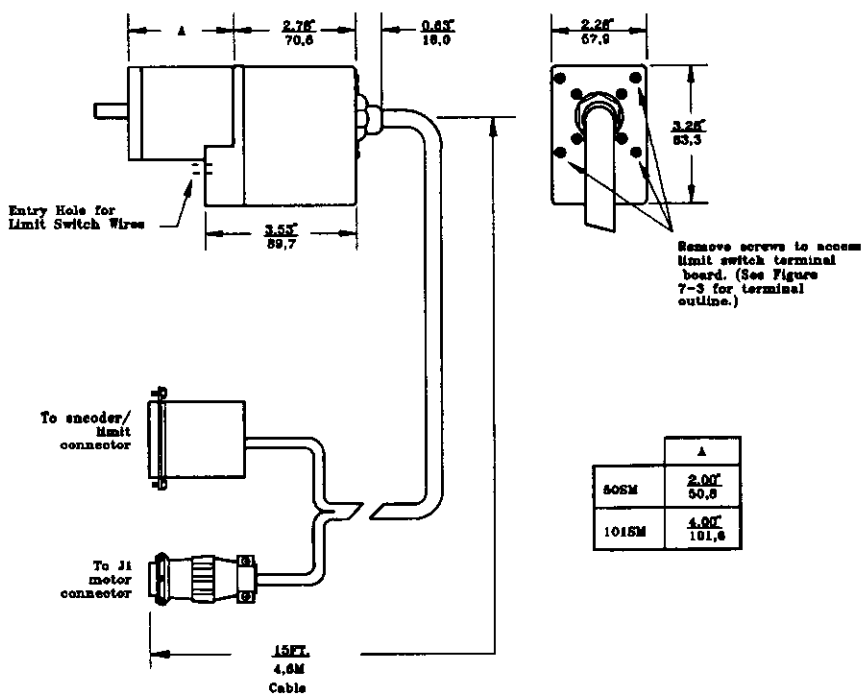


Figure 7-2: Mechanical Dimensions for Unidex 14 Stepping/Aerodrive Motors

C2E-HM - 50SM, 101SM Motors
C2E-BXXXAS 50 or 101



C3E-HM - 300SM, 310SM Motors
C3E-BXXXAS 310

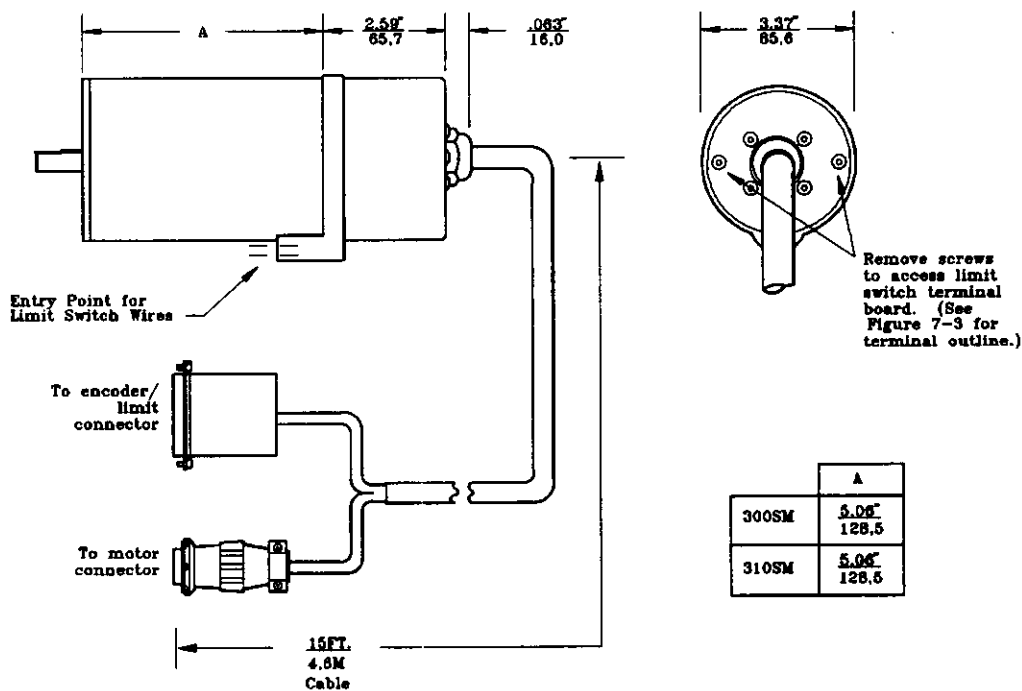
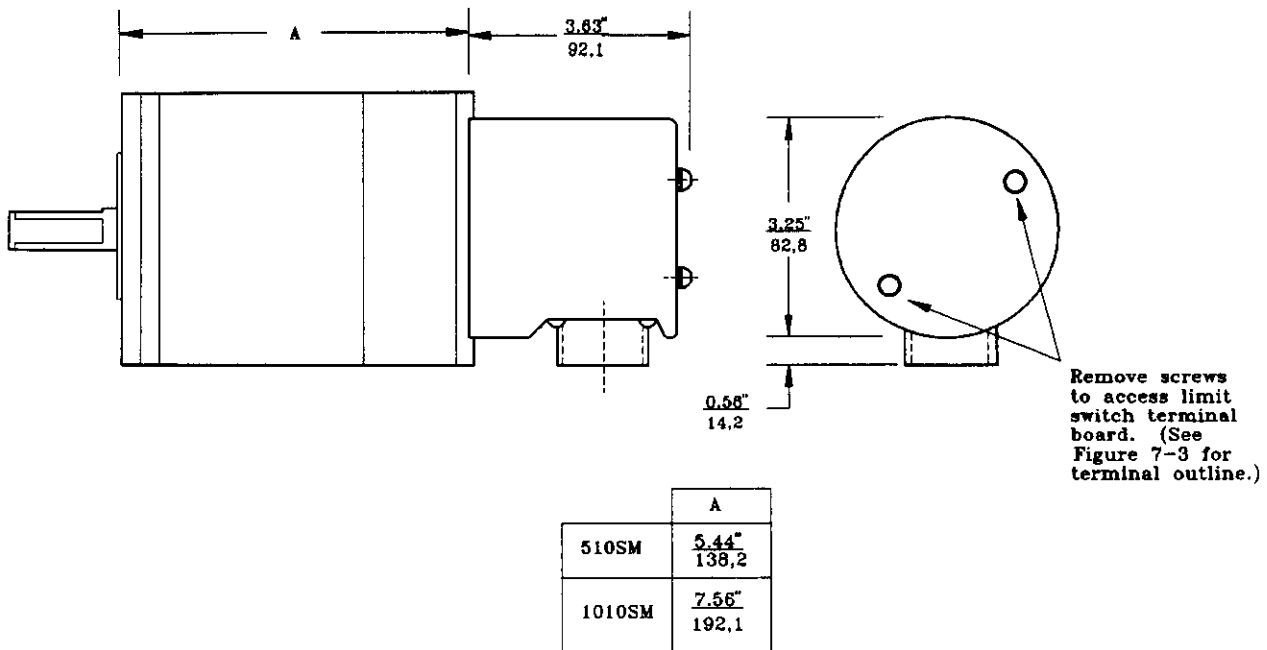
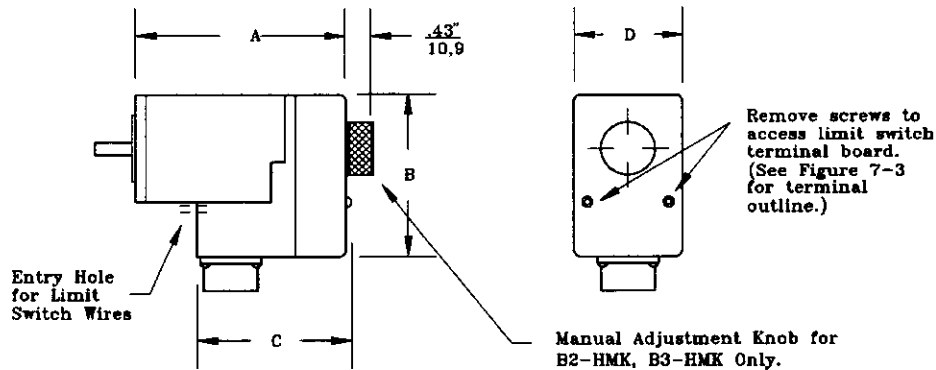


Figure 7-2: Continued

B4-HM - 510SM, 1010SM Motors
B4-EXXXAS 510 or 1010



B2-HM, B2-HMK - 50SM, 101SM Motors
B2-EXXXAS 50 or 101
B3-HM, B3-HMK - 300SM, 310SM Motors
B3 EXXXAS 310 only

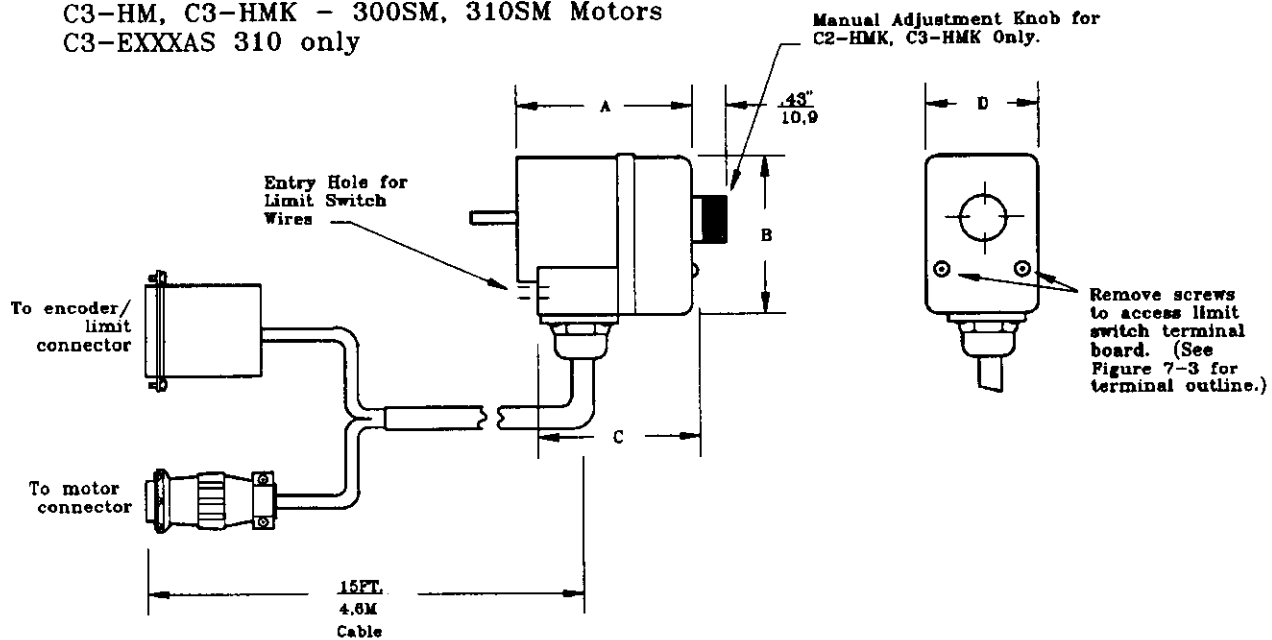


	A	B	C	D
50SM	3.32" 84,3	3.28" 83,3	3.04" 77,2	2.28" 57,9
101SM	5.32" 135,1	3.28" 83,3	3.04" 77,2	2.28" 57,9

	A	B	C	D
300SM	6.51" 165,3	4.12" 104,7	3.45" 87,6	3.40" 86,4
310SM	6.51" 165,3	4.12" 104,7	3.45" 87,6	3.40" 86,4

Figure 7-2: Continued

C2-HM, C2-HMK - 50SM, 101SM Motors
 C2-EXXXAS 50 or 101
 C3-HM, C3-HMK - 300SM, 310SM Motors
 C3-EXXXAS 310 only



	A	B	C	D
50SM	$\frac{3.32"}{84.3}$	$\frac{3.28"}{83.3}$	$\frac{3.04"}{77.2}$	$\frac{2.28"}{57.9}$
101SM	$\frac{5.32"}{135.1}$	$\frac{3.28"}{83.3}$	$\frac{3.04"}{77.2}$	$\frac{2.28"}{57.9}$

	A	B	C	D
300SM	$\frac{8.51"}{165.3}$	$\frac{4.12"}{104.7}$	$\frac{3.45"}{87.6}$	$\frac{3.40"}{86.4}$
310SM	$\frac{8.51"}{165.3}$	$\frac{4.12"}{104.7}$	$\frac{3.45"}{87.6}$	$\frac{3.40"}{86.4}$

Figure 7-2: Continued

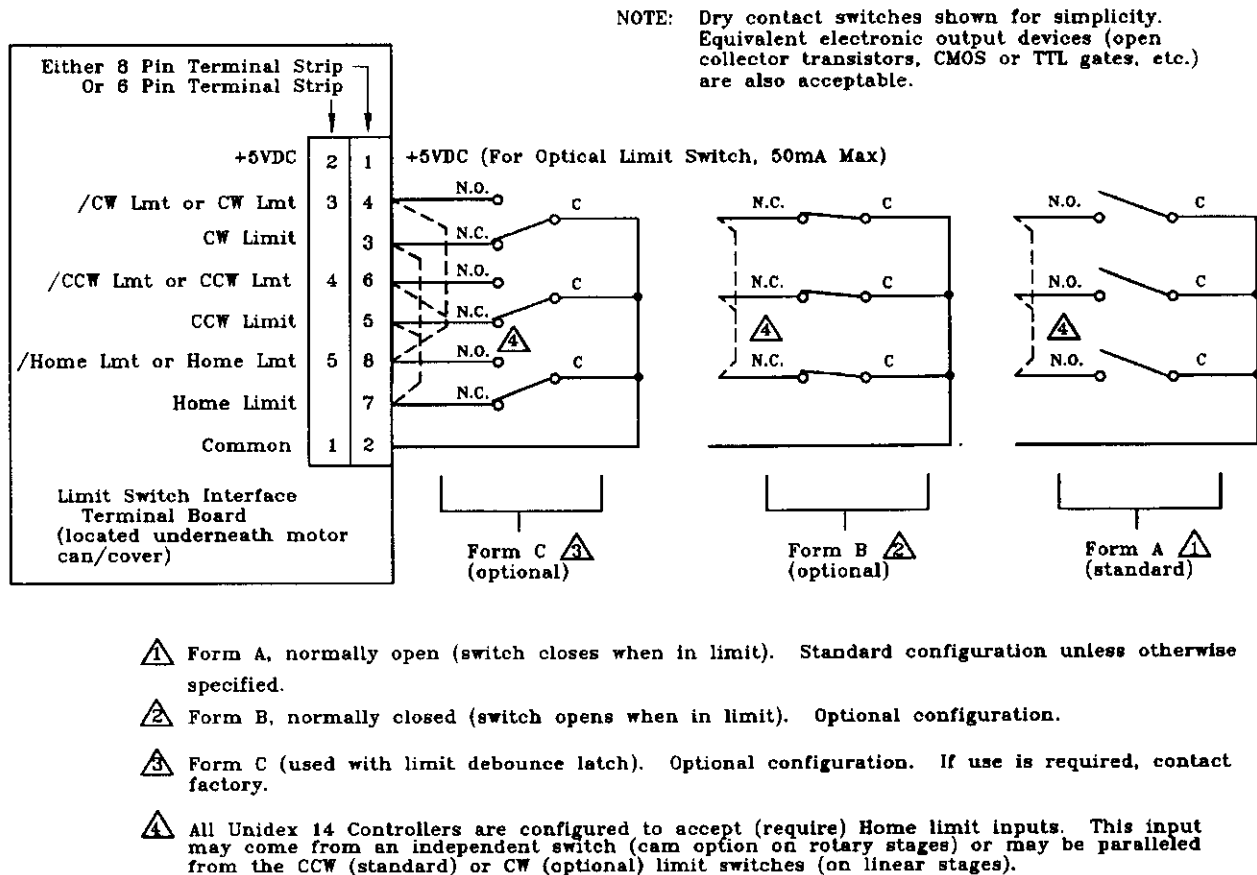


Figure 7-3: Limit Switch Terminal Board Definitions

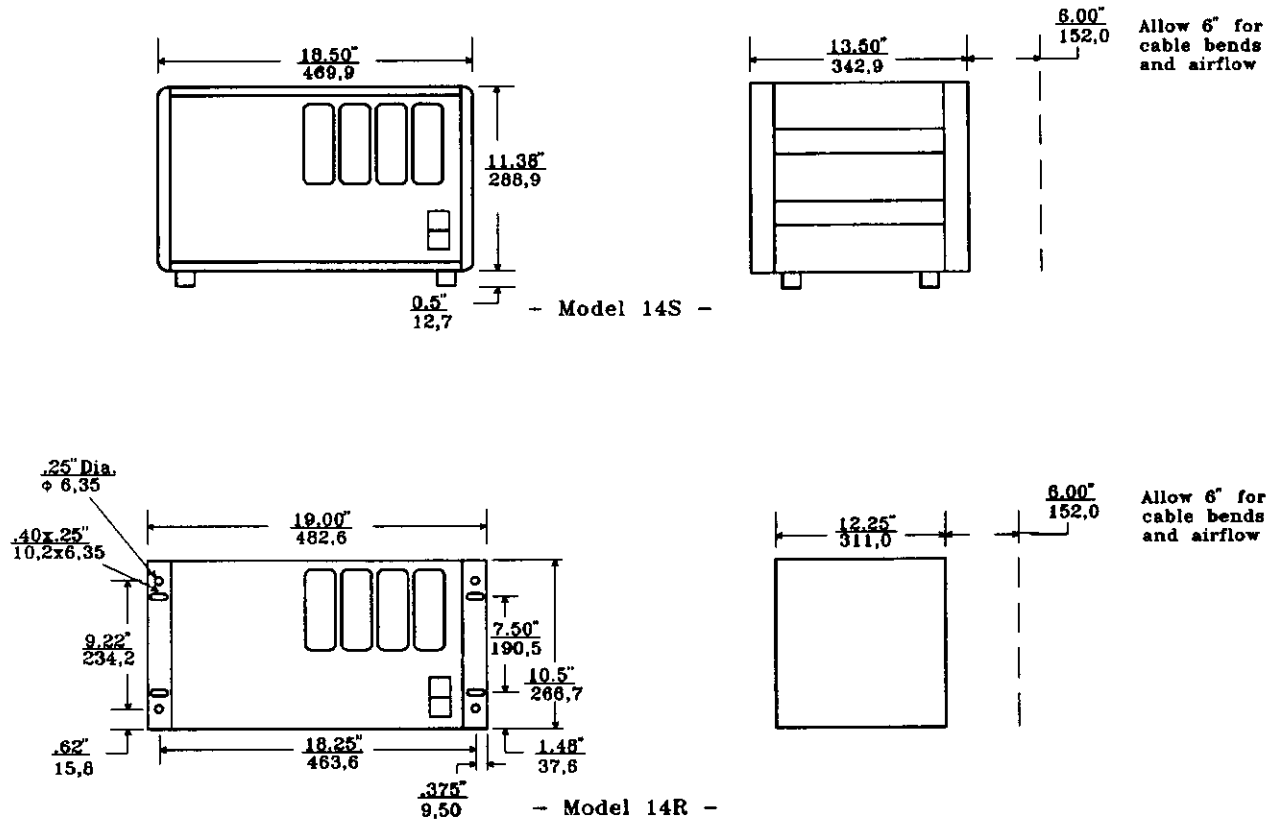


Figure 7-4: Dimensional Data for the Unidex 14 Family of Controllers

CHAPTER 8: COMMAND STRUCTURE

SECTION 8-1: INTRODUCTION

An extensive command structure is built into the Unidex 14. It includes a 200 command and parameter buffer for each axis and a command loop counter which allows multiple executions of any command string.

All of the commands are two ASCII characters and some of them expect a numerical operand to follow. These commands are identified with a '#' after the command. The operand must be terminated by a space, carriage return or semi-colon to indicate the end of the number. No terminator is required on the other commands, but may be included to improve readability. This operand must immediately follow the command with no space or separation character.

If User units are not enabled, the '#' indicates that a signed numerical input parameter is required. If User units are enabled fixed point number of the format ##.## is required. With User units enabled, distances, velocity and acceleration parameters will be input in the specified units.

Synchronized moves may be made by entering the AA command. This command performs a context switch which allows entering the commands in the format MRx#,y#,z#,t#;. Numbers are entered for each axis that is commanded to move. An axis may be skipped by entering the comma with no parameter. The command may be prematurely terminated with a ";", i.e. a move requiring only the X and Y axes would use the command MRx#,y#; followed by the GO command. Each axis programmed to move will start together upon executing the GO command. The Unidex 14 can be switched back to the unsynchronized mode by entering the desired axis command such as AX.

The AM command is provided for complex applications where the host manages multiple motion processes by a multitasking operating system. This mode shares the same instructions as the AA mode but allows starting a task while some other task involving one or many axes is active. For example the X and Y axes could be doing linear interpolation while the Z axis is making an unrelated move simultaneously.

The constant velocity contouring provides a fourth mode wherein the move parameters are predefined by entering AA then CD#, #;. The Unidex 14 will then calculate the move profile in advance and move at constant velocity in the prescribed pattern as required for machine tool and other applications. It can do linear interpolation on all 4 axes between the predefined points or it can do circular interpolation mixed with linear on two axes.

The input characters are placed in a circular queue on input then removed and interpreted. The commands are then placed in separate circular queues for each axis. As they are executed the space is reclaimed allowing the host to pass commands ahead of the moves actually being processed. The tables following each command in this section define the queue requirements for each command in each command mode for reference. The RQ command may be used to determine the actual space available at any time. The queues operate independently allowing each axis to perform separate processes simultaneously.

The synchronized modes (AA) insert special wait opcodes which allow the axes to be synchronized in this mode. When the commands are nested within loops, the queue space is not reclaimed until after the loop has been executed the programmed number of times. For loops larger than the queue space, the loop may never be completed since it cannot reclaim the queue space and cannot accept the loop terminator. The RQ command may be used to examine the remaining queue space. A control-D may clear this condition if the input character queue is not also filled since it bypasses the command interrupter.

SECTION 8-2: AXIS SPECIFICATION COMMANDS

The following commands set the context, directing the commands to the appropriate axis. They remain in effect until superseded by another command of the same type specifying a different axis. Most of the commands are placed in the appropriate command queue to be executed while others are executed immediately, allowing the return of status information in a timely way rather than when encountered in the command stream.

Queue status is provided in table format for each command. Commands that are not queued are indicated by an "immediate" status. The single axis cases are indicated by the mode reference designating the appropriate axis. The synchronized mode is indicated by the mode identifier AA. The contouring case is indicated by AA/CD for multiple axes in contour definition mode.

The syntax example also illustrates the applicable mode (AA,AX,AY,AZ,AT,AA/CD).

NOTE: All of the commands in this section are modal (remain in effect until superseded by another command of the same type) so the axis mode specifier is not required to immediately precede the command example.

AA AXIS ALL

The AA command will perform a context switch to coordinated moves.

QUEUE REQUIREMENTS

MODE

AX-AT	Immediate
AA,AM	Immediate
AA/CD	Not valid

Syntax: AA MA30, -73, ,11; GO

AM AXES MULTITASKING

The AM mode allows several tasks to be managed simultaneously. A task may be performing coordinated motion on 2 axes while a second task is performing unrelated but simultaneous motion on a third axes.

QUEUE REQUIREMENTS**MODE**

AX-AT	Immediate
AA,AM	Immediate
AA/CD	Not valid

AX AXIS X

The AX command sets the context to direct all the following commands to the X axis. This mode is the default at power up or reset.

QUEUE REQUIREMENTS**MODE**

AX-AT	Immediate
AA,AM	Immediate
AA/CD	Not valid

Syntax: AX MT-46793; GO

AY AXIS Y

The AY command sets the context to direct all the following commands to the Y axis.

QUEUE REQUIREMENTS**MODE**

AX-AT	Immediate
AA,AM	Immediate
AA/CD	Not valid

Syntax: AY MR-46793; GO

AZ AXIS Z

The AZ command sets the context to direct all the following commands to the Z axis.

QUEUE REQUIREMENTS**MODE**

AX-AT	Immediate
AA,AM	Immediate
AA/CD	Not valid

Syntax: AZ LP-91263;

AT AXIS T

The AT command sets the context to direct all the following commands to the T axis.

QUEUE REQUIREMENTS**MODE**

AX-AV	Immediate
AA,AM	Immediate
AA/CD	Not valid

Syntax: AT VL15000;

SECTION 8-3: SYSTEM CONTROL COMMANDS

These commands allow control of various system parameters and operating modes to allow the user to optimize the response of the system for his/her application needs.

EN ECHO ON

The EN command enables echoing. All commands and parameters will be echoed to the host. This mode is useful for debugging command strings from a terminal. This mode also outputs English readable error messages to the host which may be echoed to the terminal or computer to aid in debugging.

QUEUE REQUIREMENTS**MODE**

AX-AT	Immediate
AA,AM	Not valid
AA/CD	Not valid

Example: Enable echoing by the Unidex 14 so that commands are echoed and error messages are returned to the host in readable ASCII strings. This command would probably be the first command executed after turning on the system when this mode is desired.

Enter: EN

EF ECHO OFF

The EF command disables echoing from the Unidex 14 motion system. This is the default mode at power up or reset.

QUEUE REQUIREMENTS**MODE**

AX-AT	Immediate
AA,AM	Not valid
AA/CD	Not valid

Example: Stop echoing to the terminal.

Enter: EF

HH HOME HIGH

Sets the sense of the home switches to active high. This allows the use of a normally closed switch.

QUEUE REQUIREMENTS**MODE**

AX-AT	1
AA,AM	Not valid
AA/CD	Not valid

(See HL command for example)

HL HOME LOW

Sets the sense of the home switches to active low. This is the default setting on reset or power up.

QUEUE REQUIREMENTS**MODE**

AX-AT	1
AA,AM	Not valid
AA/CD	Not valid

Example: A faster home sequence may be used in applications which have a long distance to travel to reach home. The stage is moved through home at high speed with the home switch set for active high then reversed at low speed to meet the 1024 steps per second requirement of the home command.

Enter: AX VL20000 HH HM0
VL1000 HL HR0

Note: See BL# command for hardware home command.

LF LIMITS OFF

The LF command disables the sensing of the limit switches for the addressed axis, by the Unidex 14 indexer card. The drive modules themselves prevent overtravel by means of hardware sense circuit.

QUEUE REQUIREMENTS**MODE**

AX-AT	1
AA,AM	Not valid
AA/CD	Not valid

Syntax: AX LF;

LN LIMITS ON

The LN command restores the operation of the limit switches for the addressed axis. This is the default state on power up or reset.

QUEUE REQUIREMENTS**MODE**

AX-AT	1
AA,AM	Not valid
AA/CD	Not valid

Syntax: AY LN;

CN COSINE ON

Enable cosine velocity ramps, i.e. half sinusoid acceleration profiles for all axes. The cosine is not truncated in moves that do not reach full speed. See Chapter 2 for explanation of velocity profiles.

NOTE: This command should not be given while an axis is in motion or the results may be unpredictable.

QUEUE REQUIREMENTS**MODE**

AX-AT	Immediate
AA,AM	Immediate
AA/CD	Not valid

Syntax: AA CN;

PN# PARABOLIC ON

Sets all axes to truncated parabolic ramps. This acceleration profile starts at 100% of the programmed acceleration and decreases in steps of 10% of the initial acceleration down to a low of 10%. The parameter supplied selects the number of steps. It must be in the range of 3 to 10 corresponding to 70% and 10% acceleration at the peak respectively. A parameter out of this range or no parameter supplied defaults to 70% or 3 steps. Note that the parameter is the number of steps, not the acceleration values. The larger number is a lower acceleration at the peak. See Chapter 2 for explanation of velocity profiles.

NOTE: This command should not be given while an axis is in motion or the results may not be predictable.

QUEUE REQUIREMENTS**MODE**

AX-AT	Immediate
AA,AM	Immediate
AA/CD	Not valid

Syntax: AX PN;

PF PARABOLIC OFF

Restores all axes to linear acceleration and deceleration ramps (default at reset or power on).

NOTE: This command should not be given while an axis is in motion or the results may not be predictable. This is the default mode at power up or reset.

QUEUE REQUIREMENTS**MODE**

AX-AT	Immediate
AA,AM	Immediate
AA/CD	Not valid

Syntax: AX PF;

RS RESET

The RS command is a software reset which causes Unidex 14 to reinitialize. All previously entered data and commands are lost. All internal parameters are initialized to defaults and all interrupts are disabled. This command is intended for catastrophic failure recovery only. Use the KL command to reset queues or return the system to a known state.

QUEUE REQUIREMENTS**MODE**

AX-AT	Immediate
AA,AM	Immediate
AA/CD	Not valid

Syntax: AX RS

SECTION 8-4: COMMANDS FOR USER I/O

The following commands are for reading and writing the User Definable I/O.

AN AUXILIARY ON

The AN command turns on the selected auxiliary output ports. It may be used to change power level on driver modules so equipped, trigger another board's sync input or other user definable output. A parameter must be supplied for the desired axes when used in the AA mode so that the other axes are not affected. No parameter is required in the single axis mode. This is the default mode at power up or reset.

QUEUE REQUIREMENTS

MODE

AX-AT	1
AA,AM	1
AA/CD	2

Example: To turn on the Y axes auxiliary output in the single axis mode.

Enter AY AN

Example: To turn on the X and Z axes auxiliary outputs when in the AA command mode. A parameter of 0 defines the active state as off while a 1 defines the active as on. The Y axis is unchanged in this example.

Enter: AA ANO,,1;

Note: Stepper axis use the auxiliary output to control idle/run current. See PA# command.

AF AUXILIARY OFF

The AF command deactivates the selected auxiliary outputs. It may also be used to change power level on driver modules. The same parameter rules apply as the AN command.

QUEUE REQUIREMENTS**MODE**

AX-AT	1
AA,AM	1
AA/CD	2

(See AN command for example)

PA# POWER AUTOMATIC

The PA command will activate or deactivate the auxiliary outputs at the beginning of each GO command execution and complement the outputs after the GO command is executed. The auxiliary will be activated before the GO if the parameter is zero or not specified in the single axis mode. If the parameter is non-zero, the sense is reversed i.e. the auxiliary output is deactivated at the beginning of the GO command and activated at the end.

This mode need only be set once and can be deactivated by using the AN or AF command. Axes can be selectively affected in the AA mode by following the syntax as described for the AN command. The following queue requirements apply to each GO command in the command stream in the AA and single axis modes.

QUEUE REQUIREMENTS**MODE**

AX-AT	2
AA,AM	2
AA/CD	Not valid

Example: To activate the Y axis auxiliary output during a move and turn the T axes output off during a move while in the AA command mode.

Enter: AA PA,0,,1;

Note: This command should be used with stepper axis to control idle/run current. The output should be activated during motion to achieve running current. EX: AX PA.

BL# BIT LOW

The BL command sets the selected output to active low (ON). The 6 outputs are numbered 8 thru 13. Bits 12 and 13 are normally reserved for system functions. For normal operation bit 12 is set HI to enable the Unidex 14 indexer card to control the Unidex 14 Drive Chassis. Toggling bit 13 low for 9 mSec enables a CCW hardware home cycle on all four axis. See also INITSYS and home subprograms in Chapter 9. Refer to Section 4-4 for additional information on Output Signals.

OUTPUT QUEUE REQUIREMENTS**MODE**

AX-AT	2
AA,AM	2
AA/CD	Not valid

Example: To turn on output bit 10 after a move.

Enter: AX MA1000 GO BL10

BH# BIT HIGH

The BH command sets the selected output to high (OFF). The 6 outputs are numbered 8 thru 13. Bits 12 and 13 are normally reserved for system functions. (See above)

QUEUE REQUIREMENTS**MODE**

AX-AT	2
AA,AM	2
AA/CD	Not valid

(See BL command for example)

BX BIT REQUEST IN HEX

The BX command returns the state of the 8 input bits in hex format.

INPUT QUEUE REQUIREMENTS**MODE**

AX-AT	1
AA,AM	1
AA/CD	Not valid

Example: AA BX; returns LF CR $\phi\phi\phi\phi$ 1F LF CR indicating all 5 user inputs are high.

Note: Input bits 0,1,2,3,4,5,6, and 7 are user inputs. Following is a description of the alternative uses for bits 5-7:

Bit 5 HI indicates the completion of a hardware home command (See BL# instruction and "HOME" subprogram in Chapter 9). Bit 6 HI indicates a fault condition is present on a DC Servo or Aerodrive axis (See "FAULT" subprogram in Chapter 9). Bit 7 HI indicates DC Servo and Aerodrives axis have completed motion (See "INPOS" subprogram in Chapter 9).

Refer to Section 4-4 for more information on Input and Output control.

SECTION 8-5: MOVE SPECIFICATION COMMANDS

These commands allow specification of move parameters. They allow move parameters to be tailored to the users system requirements.

AC# ACCELERATION

The AC command sets the acceleration/deceleration register to the operand which follows the command. The parameter must be greater than zero and less than 8,000,000. All the following move commands for the axis being programmed will accelerate or decelerate at this rate until another AC command is entered. All acceleration registers default to 2,000,000 steps per second per second at power on or reset. The acceleration register may be modified by an ML or MT instruction in the AA or AM mode. The user must then redefine them with an AC command when returning to the single axis mode or when using move commands in the AA or AM mode which do not do interpolation such as the MA or MR commands.

MODE	<u>QUEUE REQUIREMENTS</u>		
	LINEAR	PARABOLIC	COSINE
AX-AT	4	15	15
AA,AM	4	15	15
AA/CD		Not valid	

Example: In the single axis mode, set the Y axis acceleration to 200,000 counts per second per second.

Enter: AY AC200000

Example: In the AA mode, set the acceleration of the X axis to 200,000 and the Z axis to 50,000 and leave the other axis with their previous values.

Enter: AA AC200000,,50000;

VL# VELOCITY

The VL command sets the maximum velocity register of the axis being programmed to the operand which follows the command. The operand must be greater than zero and less than or equal to 524,000 steps per second. The velocity defaults to 200,000 at power up or reset. This register controls the maximum velocity used in relative and absolute position moves except as modified by the linear interpolation instructions. If the velocity register is modified by an ML or MT instruction in the AA or AM modes, the user must redefine the velocity with a VL command when returning to the single axis mode or using a move command which does not use interpolation in the AA or AM modes.

QUEUE REQUIREMENTS

MODE	LINEAR	PARABOLIC	COSINE
AX-AT	2	13	13
AA,AM	2	13	13
AA/CD		Not valid	

Example: In the single axis mode, set the X axis velocity to 10,000 counts per second per second.

Enter: AX VL10000

Example: In the AA mode, set the peak velocity of the X axis to 5,000 and the T axis to 50,000 and leave the other axis with their previous values.

Enter: AA VL5000,,,50000;

VB# VELOCITY BASE

The VB command allows the velocity ramp to start at the specified velocity. This allows faster acceleration and the ability to pass through resonance quickly for some applications. The velocity jumps instantly to the specified velocity, then ramps as usual. The deceleration is the same in reverse. This mode is active only for linear ramps. It is ignored for cosine and parabolic ramps but not flagged as a command error. The parameter must be greater than zero and less than the programmed velocity. The base velocity defaults to zero at power up or reset.

<u>QUEUE REQUIREMENTS</u>			
MODE	LINEAR	PARABOLIC	COSINE
AX-AT	2	2	2
AA,AM	2	2	2
AA/CD		Not valid	

Syntax: AA VB200,2000,200,2000;

LP# LOAD POSITION

The LP command will immediately load the position supplied as a parameter in the absolute position register of the axis. The parameter will be loaded into the encoder position register and the (parameter times the encoder ratio) will be loaded into the position counter. If no parameter is supplied, the value of zero is used. This command turns off the position hold and interrupt on slip modes.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	2	4
AA,AM	Not valid	
AA/CD	Not valid	

Example: The following loads the X axis position register with 1000.

Enter: AX LP1000

Example: The following loads the Y axis position register with 20000 and the encoder position register with 30000 counts.

Enter: AY ER3,2 LP30000

MA# MOVE ABSOLUTE

The MA command will set up the axis to move to the absolute position supplied as a parameter. In the AA mode, an axis may remain stationary by entering the comma but omitting the parameter. The move is actually initiated by a GO command. The default value of zero is used if no parameter is supplied.

QUEUE REQUIREMENTS

MODE	LINEAR	PARABOLIC	COSINE
AX-AT	2	2	2
AA,AM	2	2	2
AA/CD		Not valid	

Example: Move the X axis to absolute position 100,000 counts with the previously entered acceleration and velocity parameters.

Enter: AX MA100000 GO

Example: Move the Y axis to absolute position 10,000 counts and the T axis to absolute position 1,000 counts. The other axis will remain in their current positions.

Enter: MA,10000,,1000; GO

MR# MOVE RELATIVE

The MR command will set up the axis to move relative from the current position at the time the move is executed. In the AA mode, an axis may remain stationary by entering the comma but omitting the parameter. The move is actually initiated by a GO command. The command syntax is the same as the MA command listed previously.

QUEUE REQUIREMENTS

MODE	LINEAR	PARABOLIC	COSINE
AX-AT	2	2	2
AA,AM	2	2	2
AA/CD		Not valid	

Syntax: AX MR-42768; GO

ML#,#, MOVE LINEAR

The ML command uses linear interpolation to perform a straight line relative move to the new location. Input parameters are relative distance for each axis in the move. Velocity and acceleration parameters of each axis may be automatically adjusted by the Unidex 14 controller to perform the linear move. If linear and single axis moves are mixed, it will be necessary to reset the velocity and acceleration parameters for the single axis move following a linear move. The parameters may have been modified by the Unidex 14 depending on the relative distances of the linear move. The ML command should be followed by a GO to start the axes together. The velocity and acceleration parameters are scaled to allow the axes to move and finish together.

QUEUE REQUIREMENTS

MODE	LINEAR	PARABOLIC	COSINE
AX-AT	6	30	30
AA,AM	6	30	30
AA/CD		Not valid	

Example: In the AA mode move the Y, Z and T axis 10,000, 100 and 1,000 counts respectively with each starting and finishing together. The other axes remain in their previous positions.

Enter: AA ML, 10000, 100, 1000; GO

MT#,#; MOVE TO

The MT command uses linear interpolation to move to the specified absolute position. The syntax is similar to the ML command. This command is invalid while in the CN mode if loops are being used. The command will become valid again after executing a ST or KL command. **When used in the contour definition mode, only the axes being used in the contour must be provided in the MT syntax.**

QUEUE REQUIREMENTS

MODE	LINEAR	PARABOLIC	COSINE
AX-AT	6	30	30
AA,AMF	6	30	30
AA/CD		4 + number of axes	

Example: In the AA mode move the X, Y and T axis to absolute positions 1,000 10,000 and 100 counts respectively with each starting and finishing together. The unused axes remain in their previous positions.

Enter: AA MT1000, 10000,,100; GO

MO MOVE ONE PULSE

The MO command will output one step pulse in the current direction (do not use the GO command). The direction may be reversed by use of the MM or MP command. This command generates the output pulse in one sample interval and thus eliminates the latency of generating a ramp with an MR1 GO command sequence.

QUEUE REQUIREMENTS

MODE	
AX-AT	1
AA,AM	Not valid
AA/CD	Not valid

Syntax: AX MO;

RM# REMAINDER

The RM command will divide the position counter by the parameter supplied and replace the position counter with the resulting remainder. The parameter must be greater than zero and less than 65000. This command is used in applications where the controller is managing the motion of a continuously rotating object. It allows the position counter to keep track of the absolute position without regard to the number of revolutions it may have rotated.

Example: A RM2000 command with a position counter of -4050 will return a position of 1950 since it is within 50 counts of rolling over at -4000, i.e. the axis is 1950 counts from the starting point.

QUEUE REQUIREMENTS
MODE

AX-AT	1
AA,AM	Not valid
AA/CD	Not valid

Syntax: AX RM2000;

SECTION 8-6: MOVE EXECUTION COMMANDS

These commands allow execution of the moves which have been previously specified.

GO GO

The GO command will initiate the move which has been previously programmed. No operand is required with the GO command.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	4	7
AA,AM	5	8
AA/CD	Not valid	

Syntax: AA MA10000,,1000; GO

GD GO and RESET DONE

The GD command may be substituted for a GO command. It will reset the done flags, then proceed with the move identical to the GO command. In the single axis mode only the done flag for the selected axis will be reset. In the AA mode all the done flags will be reset. In the AM mode the axes involved in the move will be reset. This allows the host to reset the interrupts on the axis involved in the next move without affecting other axes which may be still active. Note that this command is probably not useful in applications where commands are queued in advance since the interrupt may be reset before the host has the opportunity to service it if the GD command is waiting in the queue.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	4	7
AA,AM	5	8
AA/CD	Not valid	

Syntax: AA MA10000,,1000; GD

JG# JOG

The JG command is a constant velocity mode and will jog the axis at the velocity supplied as a parameter. The JG command will accelerate to the programmed velocity and run until stopped by an ST, KL or another JG command. The axis will then stop and wait further commands. The jog velocity may be changed by following the command with another JG command of a different velocity. The axis must be stopped before reversing directions.

QUEUE REQUIREMENTS

MODE	LINEAR	PARABOLIC	COSINE
AX-AT	2		Linear ramp
AA,AM		Not valid	
AA/CD		Not valid	

Example: Jog the motor at 100,000 steps per second then change to 35,000 steps per second when the second JG is entered then stop by decelerating to a stop.

Enter: AX JG100000 JG35000 ST

VS#,#,# VELOCITY STREAMING

The VS command will generate a pulse train without acceleration or deceleration at the rates specified. The parameters are timed in 1/1024 second sample intervals, X velocity, and Y velocity. This is a slave mode and cannot be mixed or queued with other commands. The X axis must be active since the VS commands and all parameters are inserted in the X command queue. The VS command does not require a "GO" command.

QUEUE REQUIREMENTS
MODE

AX	5
AY-AT	Not valid
AA,AM	Not valid
AA/CD	Not valid

SECTION 8-7: MOVE TERMINATION COMMANDS

The following commands allow termination of move sequences in process.

ST STOP

The ST command flushes the queue for the current axis only in the single axis mode and causes the axis to decelerate to a stop at the rate previously specified in an AC command. This command is used to stop the motor in a controlled manner from the jog mode or an unfinished GO command. This command is executed immediately. No status or position information is lost. The ST command is equivalent to the SA command in the AA mode.

QUEUE REQUIREMENTS
MODE

AX-AT	Flush +2
AA,AM	Flush +2
AA/CD	Not valid

Syntax: AX JG1000 ST

SA STOP ALL

The SA command flushes all queues and causes all axes to decelerate to a stop at the rate previously specified in an AC command. All status and position information is retained.

QUEUE REQUIREMENTS**MODE**

AX-AT	Flush +2
AA,AM	Flush +2
AA/CD	Not valid

Syntax: AX JG1000 AY JG5000 AA SA

KL KILL

The KL command will flush the command queue and terminate pulse generation of all axes immediately. It is intended for emergency termination of any program and to reset the input queues to a known state. The motor may not stop immediately even though no more pulses are delivered due to inertia of the motor rotor and load. Therefore, the position counter may not accurately reflect the true position of the motor following this command. The home sequence should be used to reestablish the position counters. A control-D is a short cut for the KL command. It bypasses the command interpreter and may work when the queue is full and the KL command can not get through the interpreter.

QUEUE REQUIREMENTS**MODE**

AX-AT	Flush +2
AA,AM	Flush +2
AA/CD	Not valid

Syntax: AX MR999999 GO KL

SECTION 8-8: LOOP CONTROL COMMANDS

These commands allow move sequences to be repeated within loops. Loops can be nested up to four levels deep on each axis.

LS# LOOP START

The LS command sets the loop counter for the axis being programmed in the single axis mode and all axes in the AA mode. The command expects a loop counter operand following the command. The commands up to the LE loop terminator will be executed the number of times specified by the operand. Loops may be nested up to four levels deep on each axis. The parameter must be less than 32,000.

QUEUE REQUIREMENTS**MODE**

AX-AT	2
AA,AM	2
AA/CD	Not valid

Example: Execute a 100,000 count relative move on the Z axis 5 times.

Enter: AZ LS5 MR100000 GO LE

Note: The first move will occur immediately after entering the GO command. The remaining 4 moves will be executed after the loop terminator LE has been entered.

Example: Execute a 100,000 count move relative on the X axis together with a 100 count move on the T axis followed by a move absolute to 100 counts on the X axis and 200 counts on the T axis four times.

Enter: LS4 ML100000,,,100; GO MT100,,,200; GO LE

LE LOOP END

The LE command terminates the most recent LS command. The axis will loop back and repeat the commands within the loop the number of times specified in the LS command.

QUEUE REQUIREMENTS**MODE**

AX-AT	2
AA,AM	2
AA/CD	Not valid

(See the LS command for an example)

WS# WHILE SYNC

The WS command will execute the commands between the WS and WD commands as a loop while the user definable I/O line is true i.e. low. When the I/O line goes high it will exit the loop and execute the commands which follow. The test is at the bottom of the loop i.e. it will always be executed at least once.

QUEUE REQUIREMENTS**MODE**

AX-AT	2
AA,AM	Not valid
AA/CD	Not valid

(See the WD command for a syntax example)

WD WHILE END

The WD command serves as the loop terminator for the WS command

QUEUE REQUIREMENTS**MODE**

AX-AT	2
AA,AM	Not valid
AA/CD	Not valid

Syntax: WS2 AY MR1000; GO WT 1; WD

WH WHILE

The WH command will execute all commands between it and the terminating WG command as a loop until terminated by a CW command. This allows repeated execution of a command sequence which can be terminated by the host after it senses an external event has occurred. These commands may not be nested but may be executed sequentially.

QUEUE REQUIREMENTS**MODE**

AX-AT	1
AA,AM	Not valid
AA/CD	Not valid

Syntax: WH AX MR1000; GO WG CW condition dependent upon time delay.

WG WHILE FLAG

The WG command serves as the terminator for the WH command

QUEUE REQUIREMENTS**MODE**

AX-AT	1
AA,AM	Not valid
AA/CD	Not valid

(See WH command for a syntax example)

CW CLEAR WHILE

The CW command breaks the WH command upon execution of the remaining commands in the loop, i.e. the current execution of the loop is finished. The WH loop is always executed at least one time since the test for the flag is at the bottom.

QUEUE REQUIREMENTS**MODE**

AX-AT	1
AA,AM	Not valid
AA/CD	Not valid

(See WH command for a syntax example)

SECTION 8-9: HOME AND INITIALIZATION CONTROL COMMANDS

These commands allow the initialization of the controller to an absolute position.

HM# HOME

The HM command will cause the current axis to find home. The position counter will be initialized to the position supplied as a parameter. The velocity should be less than 1024 counts per second to maintain accuracy of the home position loaded. The axis will not stop at home, but will initialize the position counter when the home switch becomes true and decelerate to a stop. The axis may be commanded to go home by following this command with a move absolute to the same position as specified in the HM command. The parameter defaults to zero if none is supplied.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	4	5
AA,AM	Not valid	
AA/CD	Not valid	

Example: Find the physical home position of the X axis of the stage. (NOTE: The velocity should be less than 1024 pulses per second to minimize position error for this command.) The motor runs until the home switch input is activated and then initializes the position counter to the parameter supplied. Since the motor decelerates to a stop after reaching home, it is necessary to do a MA# to the same position as specified in the home command if it is desired to physically position the device at home. The following commands will find home, initialize it to 1000 counts, then return to home. In many cases it will not be necessary to return home, only find the position and synchronize the controller to it.

Enter: AX VL1000 HM1000 MA1000 GO

Note: See BL# command for hardware home command.

HR# HOME REVERSE

The HR command will find home and initialize the position counter to the position supplied as a parameter. It behaves exactly like the HM command above except it travels in the negative direction. The parameter defaults to zero if none is supplied.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	4	5
AA,AM	Not valid	
AA/CD	Not valid	

Example: In a long stage it may be awkward to travel the full distance to home at less than 1024 pulses per second. The following will get close to home at higher speed, then refine the position at lower speed in the reverse direction.

Enter: AX VL100000 HH HM VL1000 HL HR

KM HOME AND KILL

The KM command will find home and stop generating pulses immediately, i.e. no deceleration ramp will be generated. The position counter is not affected.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	1	1
AA,AM	Not valid	
AA/CD	Not valid	

Syntax: AX VL1000 KM

KR HOME REVERSE AND KILL

The **KR** command will find home in reverse and stop generating pulses immediately, i.e. no deceleration ramp will be generated. The position counter is not affected.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	1	1
AA,AM	Not valid	
AA/CD	Not valid	

Syntax: **AX VL1000 KR**

SECTION 8-10: MOVE SYNCHRONIZATION COMMANDS

These commands allow the synchronization of moves with external events or multiple axis sequences.

ID INTERRUPT DONE

The ID command will set the done flag and interrupt the host if the interrupt has been enabled by writing the correct value to the done flag register and the status register. This allows the Unidex 14 to signal the host when a string of commands has been completed. In the AA mode, the done flag register bits will be set as each axis encounters the ID in its command stream but the done flag in the status register will not be set until all axes have executed the ID command. In the AM mode only the axes active in the most recent move will set their done flags.

QUEUE REQUIREMENTS**MODE**

AX-AT	1
AA,AM	1
AA/CD	Not valid

Example: Interrupt the host CPU after the execution of Move Absolute is finished. When the move is finished the ID command will be encountered in the command queue and will set the done flags.

Enter: AX MA100000 GO ID

II INTERRUPT INDEPENDENT

The II command allows the control to interrupt the host when each axis finishes a move. Only those axes which have been supplied a parameter in the most recent move command will cause interrupts.

QUEUE REQUIREMENTS**MODE**

AX-AT	1
AA,AM	1
AA/CD	Not valid

Example: The following command sequence would cause interrupts when the Y and T axes finish. If they do not complete at the same time, two interrupts would be generated.

Enter: MR,1000,,10000; GO II

IN# INTERRUPT NEARLY DONE

The IN command allows the control to interrupt the host when the axis or combination of axes is nearly complete. When used in an application involving probing a part after a move, the probes could start accelerating down while the stage is finishing its move, improving the overall system throughout. This command is valid in all modes. The IN command must be entered before the GO command since it is executed before the move is complete. The test is only performed during deceleration. If the IN parameter is greater than the ramp down distance, the interrupt will be generated when the control starts ramping down.

QUEUE REQUIREMENTS**MODE**

AX-AT	2
AA,AM	2
AA/CD	Not valid

Example: The following sequence would interrupt the host when the X axis is complete and the Z axis is within 10000 counts of being complete. The Y axis completion would be ignored in this example.

Enter: INO,,10000;
 MR100000, 100000; GO
 MR,,50000; GO

IC INTERRUPT CLEAR

The IC or the ASCII character Control-Y (hex 19) command will clear the done and error flags in the Unidex 14 status register and the done flag register. This command will be executed immediately and will usually be placed in the done and error handler interrupt service routine or after polling to reset the interrupt and the associated flags. The Control-Y version of this command is preferred to minimize the latency in its execution.

QUEUE REQUIREMENTS**MODE**

AX-AT	Immediate
AA,AM	Immediate
AA/CD	Not valid

Example: The IC command must be used after an ID to reset the done flags, otherwise the axis would always appear to be "done". This would normally be done in the interrupt service routine that services the DONE interrupt or the poll routine that handles the task. The control Y version of this command is preferred since it minimizes latency. The flags may be polled by an RA or RI command which will also reset the flags.

Enter: IC (or control Y, Dec. value 25, HEX 19)

SW# SYNC WAIT

The SW command allows synchronization of multiaxes moves on one or more Unidex 14's boards through the sync line, by enabling a wait until specified input bit goes high. This command causes the axes to wait for completion of other events, i.e. until the user definable I/O bit has been released before proceeding with the next command. The auxiliary line from several other axes can be wired together to cause the axis to wait until the others are all finished.

QUEUE REQUIREMENTS**MODE**

AX-AT	1
AA,AM	1
AA/CD	Not valid

Example: The following command sequence will cause the X axis to wait until the Y axis has finished its move and turned off its auxiliary output. Here the Y aux output is wired to input 0.

Enter: AY AN MR2000 GO AF
 AX SW0 MR10000 GO

WT# WAIT

The WT command will wait for the specified number of milliseconds before proceeding with the next command in the queue. In the AA mode all axes will wait. Immediate commands will not "wait". The parameter must be between 1 and 32,000.

QUEUE REQUIREMENTS**MODE**

AX-AT	3
AA,AM	3
AA/CD	Not valid

Syntax: WT5

SECTION 8-11: SYSTEM STATUS REQUEST COMMANDS

These commands allow the host to request the status of various move parameters including the status of limit and home switches.

WY WHO ARE YOU

The WY command returns the model and software revision of the Unidex 14 being addressed. A string similar to "LFCR PC38 version 1.39-4E LF CR" will be returned surrounded by carriage returns.

QUEUE REQUIREMENTS**MODE**

AX-AT	Immediate
AA,AM	Immediate
AA/CD	Not valid

Syntax: AX WY
will return: LFCR PC38 Ver 1.39-4E LFCR

RP REQUEST POSITION

The RP command returns the current position of the currently addressed axis in the single axis mode or all positions separated by commas in the AA mode. The position will be returned to the host via the data port in ASCII format. This command is not queued, i.e. the current position will be returned immediately even if the axis is in motion. The response is surrounded by LFCR sequences (shown below).

NOTE: The current position will be returned in steps and is not scaled by User units.

QUEUE REQUIREMENTS**MODE**

AX-AT	Immediate
AA,AM	Immediate
AA/CD	Not valid

Syntax: AX RP
will return (Single axis mode) LFCR 400 LFCR
Syntax: AA RP
will return (AA axis mode) LFCR 400, 20, 6100, 801 LFCR

RQ REQUEST QUEUE STATUS

The RQ command returns the number of entries available in the queue of the currently addressed axis in the single axis mode or all axes separated by commas in the AA mode. The ASCII string is surrounded by LF CR sequences. The maximum available in each command queue is 200. The numbers are fixed in length at 3 characters. When issuing an RQ command while defining a contour the available space in the contouring queue will be returned. The maximum available is 1000. The number is fixed in length at 4 characters.

QUEUE REQUIREMENTS**MODE**

AX-AT	Immediate
AA,AM	Immediate
AA/CD	Immediate

Syntax: AX RQ
will return (Single axis mode) LFCR 200 LFCR
Syntax: AARQ
will return (AA axis mode) LFCR 200, 200, 200, 200 LFCR

RA REQUEST AXIS STATUS

The RA command returns the state of the limit switches, home switches and done/error flag for the axis which is addressed. The limit flag in the hardware status register will be reset by the RA command providing that another axis is not in limit. The status is returned in the following format:

<u>CHARACTER MEANING</u>		
CHAR	SENT	DESCRIPTION
1	LF	Line feed
2	CR	Carriage return
3	CR	Carriage return
4	P	Moving in positive direction
	M	Moving in negative direction
5	D	Done (ID has been executed. This bit is reset by this command or an IC command)
	N	No ID executed yet
6	L	Axis in overtravel. Char 4 tells which direction.
	N	Not in overtravel in this direction
7	H	Home switch closed. This bit is reset when the home switch opens.
	N	Home switch not closed
8	LF	Line feed
9	CR	Carriage return
10	CR	Carriage return

QUEUE REQUIREMENTS**MODE**

AX-AT	Immediate
AA,AM	Not valid
AA/CD	Not valid

Syntax: AX RA

The following may be returned.

LFCRCR PNNN LFCRCR

RI REQUEST INTERRUPT STATUS

The RI command is an AA mode command that returns the same status information on all axes as the RA command in the single axis mode. The 4 character fields for each axis are separated by commas and the total string is surrounded by carriage returns and line feeds.

QUEUE REQUIREMENTS**MODE**

AX-AT	Not valid
AA,AM	Immediate
AA/CD	Not valid

Syntax: AA RI
will return LF CR CR PNNN, PNNN, PNNN, PNNN LFCRCR

QA QUERY AXIS

The QA command returns the status of the single addressed axis similar to the RA command except flags are not cleared.

QUEUE REQUIREMENTS**MODE**

AX-AT	Immediate
AA,AM	Not valid
AA/CD	Not valid

Syntax: AX QA
will return LFCRCR PNNN LF CR CR

QI QUERY INTERRUPT STATUS

The QI command returns the same information for all axes when in the AA mode, as the QA command does in the single axis mode. The 4 character fields for each axis are separated by commas and the string is surrounded by carriage returns and line feeds.

QUEUE REQUIREMENTS

MODE

AX-AT	Not valid
AA,AM	Immediate
AA/CD	Not valid

Syntax: AA QI
will return LF CR CR PNNN, PNNN, PNNN, PNNN LFCRCR

SECTION 8-12: COMMANDS FOR USER UNITS

The following commands allow specification of move parameters in user defined units. The controls will automatically convert all move parameters to these units once they have been initialized.

UU# USER UNITS

The UU command converts all move velocities, distances, etc. to user specified units by multiplying by the specified parameter. The multiplier parameter can be either an integer or floating point number. This command must be given in the single axis mode but will remain effective in the AA mode. The Unidex 14 defaults to user units off at power up or reset.

NOTE: The command UU-1; will reverse the programmed direction.

QUEUE REQUIREMENTS

MODE

AX-AT	Immediate
AA,AM	Not valid
AA/CD	Not valid

Syntax: AX UU1000

UF USER OFF

The UF command turns off the user units. This command is equivalent to and preferred over UU1 since it turns off the mode thus minimizing unnecessary overhead.

QUEUE REQUIREMENTS**MODE**

AX-AT	Immediate
AA,AM	Not valid
AA/CD	Not valid

Syntax: AX UF

SECTION 8-13: POSITION MAINTENANCE COMMANDS

ER#,# ENCODER RATIO

The ER command allows specification of encoder ratio by entering encoder counts followed by motor counts for position maintenance mode. This ratio must be equal to one i.e. encoder counts must match motor counts when slip detection is enabled. All distance, velocity and acceleration parameters are input in encoder counts when this mode is enabled. The correct number of motor counts are generated, while this mode is enabled the user need only be concerned with encoder counts. This mode can be combined with user units allowing units such as inches or revolutions to be specified in encoder counts. All parameters are then input in the user units which have been defined. The ratio defaults to 1 on power up or reset.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	Not valid	1
AA,AM	Not valid	
AA/CD	Not valid	

Syntax: AX ER2000, 4000;

HV# HOLD VELOCITY

The HV command specifies maximum position hold correction velocity. This is the peak velocity which will be used while making position corrections.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	Not valid	2
AA,AM	Not valid	
AA/CD	Not valid	

Syntax: AX HV2000;

HG# HOLD GAIN

The HG allows the user to specify the position hold gain parameter. The gain parameter is multiplied by the position error in determining the velocity during correction. The parameter must be between 1 and 32,000. The parameter should be set experimentally by increasing it until the system is unstable then reducing it slightly below the threshold of stability.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	Not valid	2
AA,AM	Not valid	
AA/CD	Not valid	

Syntax: AX HG30;

NOTE: Do not use this command if a DC Brush or Aerodrive system is being used for a given axis.

HD# HOLD DEADBAND

The HD command specifies deadband counts for position hold. If the motor is within this limit, it is considered in position and no further correction will be made. This parameter interacts with the HG command, i.e. a larger deadband will allow a larger gain parameter in many applications. A parameter of zero is allowed.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	Not valid	2
AA,AM	Not valid	
AA/CD	Not valid	

(See HN for an example)

NOTE: Do not use this command if a DC Brush or Aerodrive system is being used for a given axis.

HF HOLD OFF

The HF command disables position hold, stall detection and tracking modes. This is the default mode at power up or reset.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	Not valid	2
AA,AM	Not valid	
AA/CD	Not valid	

Syntax: AX HF

HN HOLD ON

The HN command enables position correction after a move and activates the HV, HG and HD commands. Hold and slip detection are disabled if an LP, SA, ST or KL command is entered.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	Not valid	2
AA,AM	Not valid	
AA/CD	Not valid	

Example: The following commands could be used to set up the position correction mode. This sequence sets up a move velocity of 100,000 steps per second and an acceleration of 500,000 steps per second. The position correction velocity is set for 50,000 steps per second, a dead band of 10 steps and correction gain of 2,000. The correction is then enabled. A 200,000 step move is performed, then that position is maintained within the 10 step deadband until commanded to a new position.

Enter: AX VL100000 AC500000
 HV50000 HD10 HG2000 HN
 MR200000 GO

NOTE: Do not use this command if a DC Brush or Aerodrive system is being used for a given axis.

SECTION 8-14: SLIP AND STALL DETECTION COMMANDS

ES# ENCODER SLIP TOLERANCE

The ES command parameter specifies tolerance before slip or stall is flagged in the status register. The mode must be turned on with an IS command and off with an HF command.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	Not valid	2
AA,AM	Not valid	
AA/CD	Not valid	

Syntax: AX ES3;IS

NOTE: The Encoder Ratio must be equal to zero.

IS INTERRUPT ON SLIP

The IS command enables the Unidex 14 to interrupt the host upon a slip or stall detection if the appropriate bit has been set in the interrupt control register and the Encoder ratio is set to one.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	Not valid	1
AA,AM	Not valid	
AA/CD	Not valid	

Syntax: AX IS

SECTION 8-15: ENCODER TRACKING COMMANDS

ET ENCODER TRACKING

The ET command energizes the tracking mode. The axis will track it's encoder input thus allowing one axis to follow the activity of another. No acceleration or deceleration ramps are generated. The axis will duplicate the encoder input. The ER command allows the user to scale the motor's movements relative to the encoder.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	Not valid	2
AA,AM	Not valid	
AA/CD	Not valid	

Syntax: AX ET

HF HOLD OFF

The HF command disables position hold, stall detection and tracking modes.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	Not valid	2
AA,AM	Not valid	
AA/CD	Not valid	

Syntax: AX HF

SECTION 8-16: ENCODER STATUS REQUEST COMMANDS

EA ENCODER STATUS

The EA command returns encoder status of currently addressed axis in the following format:

<u>EA COMMAND RESPONSE DESCRIPTION</u>			
CHAR	SENT		DESCRIPTION
1	LF		Line feed
2	CR		Carriage return
3	CR		Carriage return
4	E	Either Character	Slip detection enabled
	D		Slip detection disabled
5	E	Either Character	Position maintenance enabled
	D		Position maintenance disabled
6	S	Either Character	Slip or stall detected (reset by execution of EA command)
	N		No slip or stall detected
7	P	Either Character	Position Maintenance within deadband
	N		Position not within deadband
8	H	Either Character	Axis is home
	N		Axis is not home
9	N		Unused/reserved
10	LF		Line feed
11	CR		Carriage return
12	CR		Carriage return

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	Not valid	Immediate
AA,AM	Not valid	
AA/CD	Not valid	

Syntax: AX EA

The following may be returned:

LFCRCR DDNNNN LFCRCR

RE REQUEST ENCODER POSITION

The RE command returns current encoder position of the addressed axis, in encoder counts. The ASCII string is surrounded by carriage returns. The position is not scaled by User Units.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	Not valid	2
AA,AM	Not valid	
AA/CD	Not valid	

Syntax: AX RE

The following may be returned:

LFCR -45678 LFCR

SECTION 8-17: VELOCITY STAIRCASE COMMANDS

The following commands describe the velocity staircase mode. This mode is useful in applications requiring a change in velocity at a prescribed position without stopping.

MP MOVE POSITIVE

The MP command sets the direction logic to move in the positive direction.

QUEUE REQUIREMENTS**MODE**

AX-AT	1
AA,AM	Not valid
AA/CD	Not valid

Syntax: AX MP MO

MM MOVE MINUS

The MM command sets the direction logic to move in the negative direction.

QUEUE REQUIREMENTS**MODE**

AX-AT	1
AA,AM	Not valid
AA/CD	Not valid

Syntax: AX MM MO

MV#,# MOVE VELOCITY

The MV command causes the motor to run to the new **ABSOLUTE** position (parameter 1) at the new velocity (parameter 2). When the destination is reached control will be passed to the next command which should be another MV command or an SP command. If the command is not received in time the controller will continue to move at the specified velocity. Note that this is a slave mode and it is the responsibility of the user to provide the commands in time. They may be queued ahead of time.

If a new MV command is sent after the controller has already passed the destination specified in the command, the controller will continue to move at the old velocity. Any number of steps can be specified in this manner with both acceleration and deceleration. The controller will not reverse direction if the position has already passed, but will behave as explained above. Thus the direction of the move must be specified before starting the move with the MP or MM commands above. All destinations must be in absolute position, no position relative moves are allowed due to the nature of these commands.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	4	5
AA,AM	Not valid	
AA/CD	Not valid	

Example Generate a velocity staircase with the breakpoints given in absolute position.

Enter: MP
 MV10000,30000
 MV20000,50000
 MV30000,10000
 SP35000

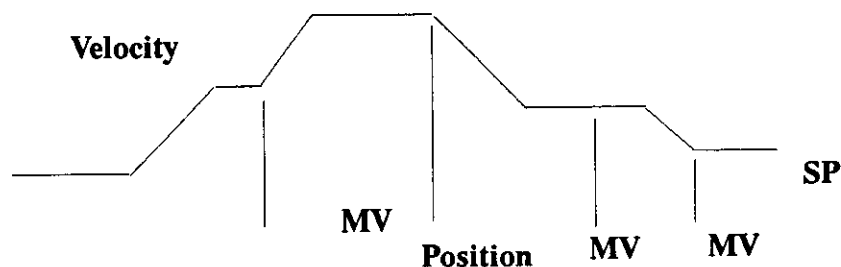


Figure 8-1: Velocity Staircase Profile

SP# STOP AT POSITION

The SP command will cause the axis to stop at the specified position. The controller will attempt to stop at the specified destination. If there is insufficient distance to stop at the previously specified deceleration when the command is received, the controller will stop as soon as possible at that deceleration. This command is not compatible with the JG command and will probably overshoot the desired position if used with it.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	3	4
AA,AM	Not valid	
AA/CD	Not valid	

Syntax: AX MR5000; GO SP3000;

FP# FORCE POSITION

The FP command will flush the command queue and attempt to stop at the specified position. The axis will overshoot if there is insufficient distance left to stop at the programmed acceleration. This command is not compatible with the JG command and will probably overshoot the desired position if used with it.

QUEUE REQUIREMENTS

MODE	NO ENCODER	ENCODER
AX-AT	Flush +4	Flush +4
AA,AM	Not valid	
AA/CD	Not valid	

Syntax: AX MR5000; GO FP3000;

SECTION 8-18: CONSTANT VELOCITY CONTOURING

The following commands are available in the Unidex 14 for the constant velocity contouring option:

CD#,#; CONTOUR DEFINE

The CD command enters contour definition mode. It allows entry of commands for contouring mode. Commands are queued for execution by the CX command. The parameters define the axes for which the contour is defined and the starting position of the contour in absolute units. The contour may be defined on all 4 axes if circular interpolation is not used or 2 axes with circular mixed with linear interpolation. Attempting to do circular interpolation in a contour which is being defined for more than 2 axes will be flagged as a command error. This command is executed in the AA mode. The contouring axes must be at positions which allow them to reach the specified contouring velocity by the specified position when the contour is executed. If the actual position of the stage is equal to the starting position as defined by the CD command, the stage will jump to the contouring velocity with no ramp up. This could cause the stage to stall if it is not able to accelerate at this high rate. It is recommended that some ramp up distance be allowed. The distance required may be calculated from the equations in Section 1.

QUEUE REQUIREMENTS**MODE**

AX-AT	Not valid
AA,AM	0
AA/CD	Not valid

Example: The following demonstrates cutting a hole with a 10,000 count radius using constant velocity contouring and circular interpolation. The contouring velocity is set to 1000 pulses per second. A contour is then defined beginning at coordinates 0,0 on the Z and T axes. The auxiliary output of the Y axis is turned on, which could turn on the cutting torch or laser starting the cut at the center of the circle. A half circle is cut from the center to the outside of the hold positioning the cutting tool at the start of the desired hole. The hole is then cut, the torch turned off, the stage stopped and the definition is complete. The stage is then positioned and the hole cut with the CX command. **Note that no commas are provided in the MT and CR commands for the inactive X and Y axes within the contour definition.** The AN and AF commands must have commas for all axes since they can all be addressed from within the contour definition.

Enter: CV1000 CD,,0,0;
AN,0; CR0,5000,3.1415926
CR0,0,6.2831853
AF,0; MT-10,10000
CE
MT,, -1000,0; GO CX

CE CONTOUR END

End of the contour sequence, terminate the CD mode or ramp to a stop and exit to AA command mode when executed. The end of the contour should contain at least a short linear segment just prior to the CE command to initialize the parameters for the deceleration of the stage.

QUEUE REQUIREMENTS**MODE**

AX-AT	Not valid
AA,AM	Not valid
AA/CD	1

(See CD command for an example)

CK CONTOUR END AND KILL

The CK command will end the contour with no ramp down, i.e. the pulses will stop abruptly. This command should be used with caution to prevent the stage from missing steps or losing its correct position.

QUEUE REQUIREMENTS**MODE**

AX-AS	Not valid
AA,AM	Not valid
AA/CD	1

Enter: CV1000 CD,,0,0;
 AN,0; CR0,5000,3.1415926
 CR0,0,6.2831853
 AF,0; MT-10,10000
 CK
 MT,,-1000,0; GO CX

CR#,#,# CIRCULAR INTERPOLATION

Move in a circular pattern from the entry position. The first two parameters are the center of the circle in absolute units and the third parameter is the distance to move in radians. The distance parameter should be supplied to seven significant digits if a full circle is to be generated.

QUEUE REQUIREMENTS**MODE**

AX-AT	Not valid
AA,AM	Not valid
AA/CD	8

(See CD command for an example)

CV# CONTOUR VELOCITY

Allows specification of contouring velocity. This command is executed from the AA mode before contour definition. A contour defined by a CD command can not be executed if followed by a CV command. The contour velocity defaults to 1000 on power up or reset.

QUEUE REQUIREMENTS**MODE**

AX-AT	Not valid
AA,AM	Immediate
AA/CD	Not valid

(See CD command for an example)

CX CONTOUR EXECUTE

Execute the previously entered contour sequence. The stage must be positioned such that it can accelerate to speed by the absolute position specified by the CD command it is executing and must be traveling in the proper direction. Once a contour is defined it may be executed at any time by executing a CX command until it is replaced by another contour definition. The CX command can not be placed within a loop or while construct.

QUEUE REQUIREMENTS**MODE**

AX-AT	Not valid
AA,AM	6
AA/CD	Not valid

The Unidex 14 will attempt to generate any profile which it is asked to do. It is the responsibility of the host to be sure the acceleration required when generating a circle or any other change in direction is possible within the mechanical constraints of the system. All corners must be defined by arcs and tangents to those arcs else the change in direction will be instantaneous and generate very large accelerations. The arc radius must be chosen so that the acceleration constraints of the system are met.

Note: **A new contour may be defined, but cannot be executed until the previous contour has finished execution.**

(See CD command for an example)

SECTION 8-19: COMMAND SUMMARY

The following commands are included in the Unidex 14 family of motor controllers. The '#' indicates a signed integer input parameter or a signed fixed point number of the format ##.# when user units are enabled. With User Units enabled, distances, velocity and acceleration parameters may be input in inches, revolutions, etc.

SUMMARY OF COMMANDS

PAGE NO.	COMMAND	COMMAND DESCRIPTION
8-3	AA	Any following commands are for the AA (All Axes) mode
8-16	AC#	Acceleration, set acceleration//deceleration register
8-13	AF	Auxiliary off
8-4	AM	Axis multitasking mode
8-12	AN	Auxiliary on
8-5	AT	Any following commands are for the T axis
8-5	AX	Any following commands are for the X axis (default on reset)
8-5	AY	Any following commands are for the Y axis
8-5	AZ	Any following commands are for the Z axis
8-14	BH#	Set selected I/O bit high (off)
8-14	BL#	Set selected I/O bit low (on)
8-15	BX	Return bit status in hex format
8-55	CD#,#;	Define a contour
8-57	CE	Contour End, exits to AA command mode
8-57	CK	Contour End, immediate pulse stop
8-9	CN	Cosine on, enables cosine velocity profiles
8-58	CR#,#,#	Circular interpolation, move in a circle
8-58	CV#	Contouring velocity, definition
8-30	CW	Clear while flag, i.e. terminate WH/WG loop
8-59	CX	Contour execute
8-50	EA	Encoder status, return encoder status of currently addressed axis
8-7	EF	Echo off, turn off echo to host (default on power on)
8-6	EN	Echo on, turn on echo to host

SUMMARY OF COMMANDS

PAGE NO.	COMMAND	COMMAND DESCRIPTION
8-44	ER#,#	Encoder ratio, sets encoder count to motor count ratio
8-48	ES#	Encoder slip tolerance, sets tolerance before slip or stall is flagged
8-49	ET	Encoder tracking, sets encoder tracking mode
8-54	FP#	Force position, will flush queue and attempt to stop at specified position
8-24	GD	Go and reset done flags
8-23	GO	Go command, starts execution of motion
8-46	HD#	Hold deadband, specifies dead band tolerance for position hold
8-46/8-49	HF	Hold off, disable position hold, slip detection and tracking modes
8-45	HG#	Hold gain, specifies position hold gain parameter
8-7	HH	Home high, home switches are active high
8-8	HL	Home low, home switches are active low
8-31	HM#	Home, find home and initialize the position counter
8-47	HN	Hold on, enables position correction after move
8-32	HR#	Home reverse, find home in reverse direction and initialize position counter
8-45	HV#	Hold velocity, specifies maximum position hold correction velocity
8-37	IC	Interrupt clear, clear done interrupt status and error flags
8-34	ID	Interrupt host when done and set done flag
8-35	II	Interrupt independent
8-36	IN#	Interrupt when nearly done
8-48	IS	Interrupt slip, interrupts host on slip or stall detection
8-24	JG#	Jog command, motor will run at specified velocity until a new velocity command is sent or it is stopped by a stop or kill command

SUMMARY OF COMMANDS

PAGE NO.	COMMAND	COMMAND DESCRIPTION
8-26	KL	Kill; flush queue and terminate pulse generation immediately on all axes without decelerating
8-32	KM	Home and kill pulse generation
8-33	KR	Home in reverse and kill pulse generation
8-28	LE	Loop end, terminates most recent LS command
8-8	LF	Disable limit switches for selected axis
8-9	LN	Enable limit switches for selected axis
8-10	LP#	Load position, loads position counter with parameter
8-27	LS#	Loop start, set loop counter, from 1 to 32000 loops; (may be nested to 4 levels)
8-20	MA#	Move absolute, move to absolute position
8-21	ML#,#;	Move linear, move specified distance relative from current position
8-52	MM	Move minus, sets minus direction for MV type move
8-22	MO	Move one pulse in current direction
8-52	MP	Move plus, sets positive direction for MF type move
8-20	MR#	Move relative, move specified distance from current position
8-22	MT#,#;	Move to, move to specified position
8-53	MV#,#	Move velocity, move to first parameter (absolute position) at second parameter velocity without stopping at end of move
8-13	PA	Power automatic, turn power on before each move and off after the move
8-10	PF	Parabolic off, disable parabolic ramps, i.e. linear ramps will be generated
8-10	PN	Parabolic on, enable parabolic ramps
8-42	QA	Query status of switches and flags for addressed axis without affecting flags
8-43	QI	Query status of switches and flags on all axes without affecting flags
8-41	RA	Return status of switches and flags and reset flags

SUMMARY OF COMMANDS

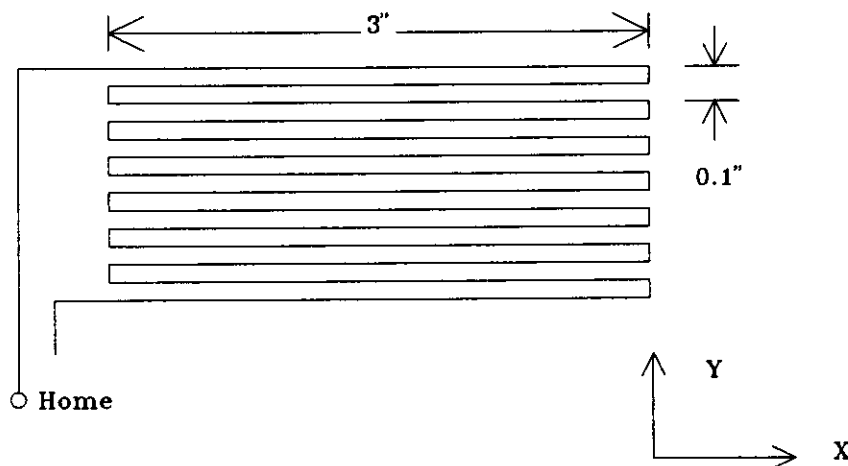
PAGE NO.	COMMAND	COMMAND DESCRIPTION
8-51	RE	Request encoder position, returns current encoder position
8-42	RI	Return status of switches and flags for all axes and reset flags
8-23	RM#	Return remainder of position divided by parameter in position counter
8-39	RP	Request position, returns current position
8-40	RQ	Request queue status, return number of queue entries available
8-11	RS	Software reset of Unidex 14
8-26	SA	Stop all, flushes queue and stops all axes with deceleration
8-54	SP#	Stop at position, will stop at specified position if possible after all commands have been executed
8-25	ST	Stop, flush queue and decelerate to stop
8-38	SW#	Sync wait, wait for the specified input to be HI.
8-44	UF	User units off, turn off user unit translation
8-43	UU#	User units, multiply acceleration, velocity and distance parameters by specified parameter
8-18	VB#	Base velocity, sets base velocity
8-17	VL#	Sets maximum velocity to be used in profile
8-25	VS#,#,#	Velocity stream, slave velocity mode for profiling
8-29	WD	While end, WH loop terminator
8-30	WG	Terminates WH loop
8-29	WH	While, execute all commands until WG loop terminator until flag cleared by CW command
8-28	WS#	While sync, execute while sync is true (low)
8-38	WT#	Wait, wait for specified number of milliseconds
8-39	WY	Who are you, returns software revision

SECTION 8-20: APPLICATION PROGRAM EXAMPLES

The following program examples illustrate the versatility of the Unix 14 programming language. These programs generate common X-Y contours that are found in many applications. The programs also show how easily repeat loops and I/O control can be incorporated into programs.

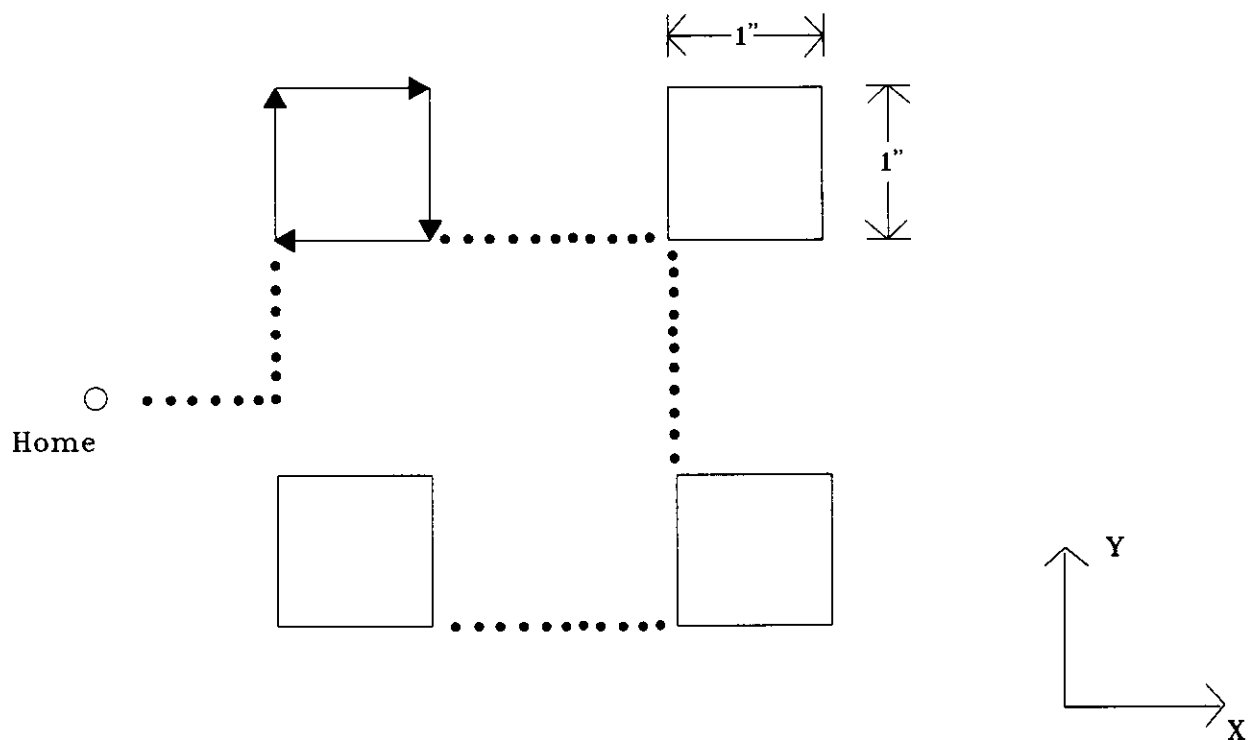
8-20-1: RASTER SCAN

BL8	Bit 8 low (output OFF)
AX UU12700	Set X axis to inches
AC25 VL.1	Set accel to 25"/sec ² and velocity to 0.1"/sec
LF HM0 MA0 GO	Turn off limits and go home
AY UU12700	Set Y axis to inches
AC25 VL.1	Set accel to 25"/sec ² and velocity to 0.1"/sec
LF HM0 MA0 GO	Turn off limits and go home
AX LN	Turn ON X axis limits
AY LN	Turn ON Y axis limits
AA	All axes motion
VL2,6;	Set velocity to 2"/sec for X and 6"/sec for Y
BH8	Bit 8 high (output ON)
MR,1.5; GO	Move Y axis 1.5"
MR0.1; GO	Move X axis 0.1"
LS7	Loop start 7 times
MR3; GO	Move X axis 3"
MR,-0.1; GO	Move Y axis -0.1"
MR-3; GO	Move X axis -3"
MR,-0.1;GO	Move Y axis -0.1"
LE	Repeat loop end
BL8	Bit low (output OFF)



8-20-2: REPEATING PATTERN AND PROGRAMMED OUTPUTS

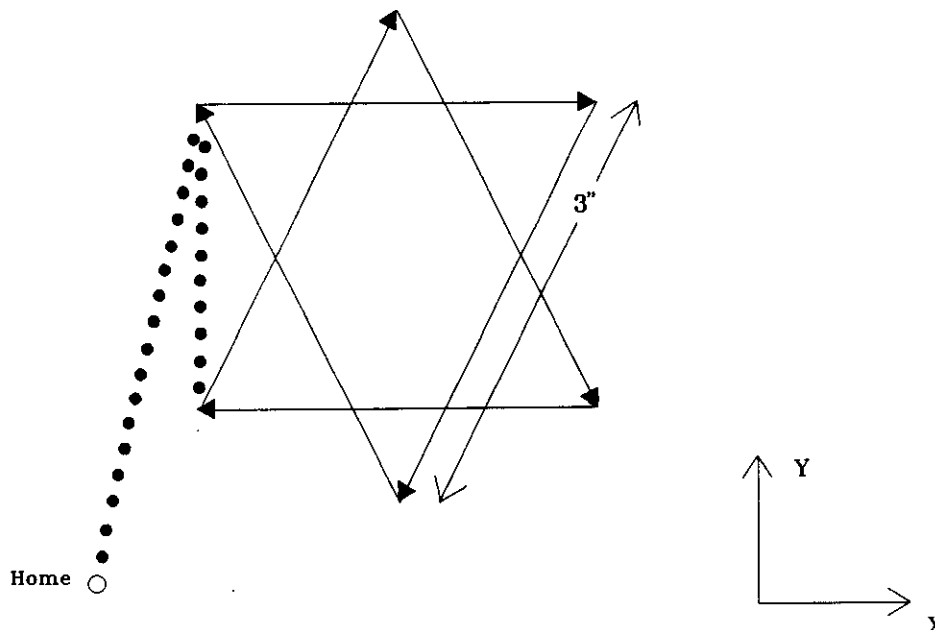
BL8	Bit 8 set low (output OFF)
AX UU12700	Set X axis to inches
AC25 VL.1	Set accel to 25"/sec ² and velocity to 0.1"sec
LF HM0 MA0 GO	Turn off limits and go home
AY UU12700	Set Y axis to inches
AC25 VL.1	Set accel to 25"/sec ² and velocity to 0.1"sec
LF HM0 MA0 GO	Turn off limits and go home
AX LN	Turn ON X axis limits
AY LN	Turn ON Y axis limits
AA	All axes motion
VL5,5;	Set velocity to 5"/sec for X and Y axis
MR0.5; GO	Move X axis 0.5"
MR,0.5; GO	Move Y axis 0.5"
BH8	Bit 8 high (output ON)
MR,1; GO	Move Y axis 1"
MR1; GO	Move X axis 1"
MR,-1; GO	Move Y
MR-1; GO	Move X axis -1"
BL8;	Bit 8 low (output OFF)
MR2; GO	
BH8	
MR,1; GO	
MR1; GO	Move X axis 2"
MR,-1; GO	(Repeat square)
MR-1; GO	
BL8	
MR,-2; GO	
BH8	
MR,1; GO	
MR1; GO	Move Y axis -2"
MR,-1; GO	(Repeat square)
MR-1; GO	
BL8	
MR-2; GO	
BH8	
MR,1; GO	
MR1; GO	Move X axis -2"
MR,-1; GO	(Repeat square)
MR-1; GO	
BL8	



REPEATING PATTERN

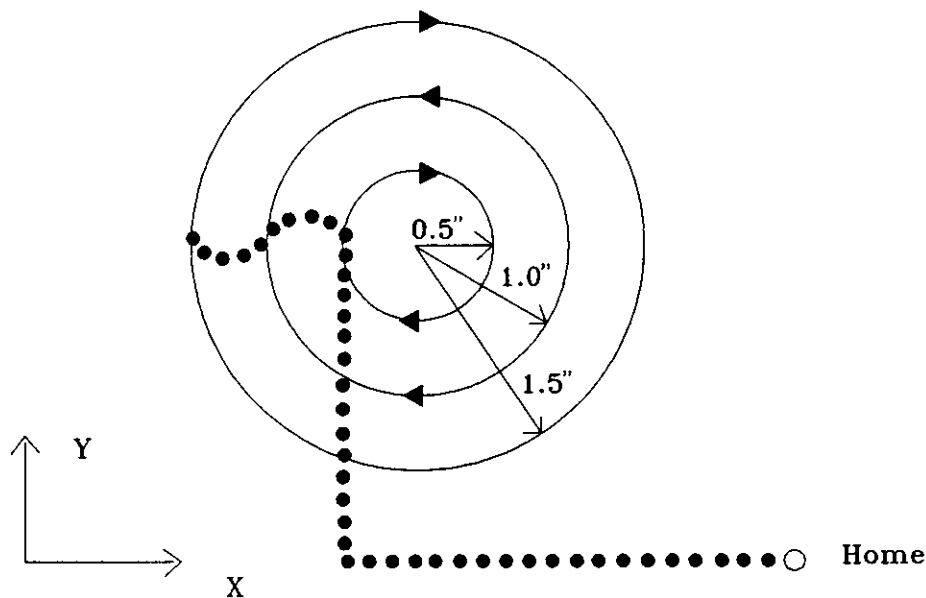
8-20-3: LINEAR INTERPOLATION - STAR

BL8	Bit 8 set low (output OFF)
AX UU12700	Set X axis to inches
AC25 VL.1	Set accel to 25"/sec ² and velocity to 0.1"/sec
LF HM0 MA0 GO	Turn off limits and go home
AY UU12700	Set Y axis to inches
AC25 VL.1	Set accel to 25"/sec ² and velocity to 0.1"/sec
LF HM0 MA0 GO	Turn off limits and go home
AX LN	Turn ON X axis limits
AY LN	Turn ON Y axis limits
AA	All axes motion
VL4.9,4.9	Set X and Y velocity to 4.9"/sec.
ML,5,3; GO	Interpolate X axis 0.5" and Y axis 3" at 5"
BH8	Bit high (output ON)
AC25,25; VL5;	Reset X,Y accel and X velocity
ML3; GO	Move X axis 3"
VL4.3,4.3	Set X and Y velocity to 4.3"/sec.
ML-1.5,-2.6; GO	Interpolate X axis -1.5" and Y axis -2.6" at 5"/sec.
BL8	Bit 8 low (output OFF)
AC25,25; VL,5;	Reset X,Y accel and velocity
ML,-1.75; GO	Move Y axis -1.75"
BH8	Bit 8 high (output ON)
VL4.3,4.3;	Set X and Y velocity to 4.3"/sec.
ML1.5,2.6; GO	Interpolate X axis 1.5" and Y axis 2.6" at 5"/sec.
AC25,25; VL4.3,4.3;	Reset X,Y accel and velocity
ML1.5,-2.6; GO	Interpolate X axis 1.5" and Y axis -2.6" at 5"/sec.
AC25; VL5;	Reset X axis accel and velocity
ML-3; GO	Move X axis -3"
BL8	Bit 8 low (output OFF)



8-20-4: THREE CONCENTRIC CIRCLES

AA AF0;	X axis aux output OFF
AX UU12700	Set X axis to inches
AC25 VL.1	Set accel to 25"/sec ² and velocity to 0.1"/sec
LF HM0 MA0 GO	Turn off limits and go home
AY UU12700	Set Y axis to inches
AC25 VL.1	Set accel to 25"/sec ² and velocity to 0.1"/sec
LF HM0 MA0 GO	Turn off limits and go home
AX LN	Turn ON X axis limits
AY LN	Turn ON Y axis limits
AA	All axes motion
VL5,5;	Set X,Y velocity to 5"/sec.
CV2	Set contouring velocity to 2"/sec.
CD-1.5,2;	Begin contour at position -1.5,2
AN0;	X axis aux output ON
CR-1,2,-6.2831853	Clockwise circle centered at position -1,2
AF0;	X axis aux output OFF
CR-1.75,2,3.1415926	Counterclockwise semicircle centered at -1.75,2
AN0;	X axis aux output ON
CR-1,2,6.2831853	Counterclockwise circle centered at -1,2
AF0;	X axis aux output OFF
CR-2.25,2,-3.14155826	Clockwise semicircle centered at -2.25,2
AN0;	X axis aux output ON
CR-1,2,-6.2831853	Clockwise circle centered at -1,2
AF0;	X axis aux output OFF
CE	End contour
MA-1.5; GO	Move X axis -1.5"
MA,1.9; GO	Move Y axis 1.9"
CX	Execute contour
MT,1.2; GO	Move Y axis 2.1"



CHAPTER 9: HOST SOFTWARE

SECTION 9-1: DEMONSTRATION SOFTWARE

The following is a print-out of the programs available at the time that this manual was printed that are provided on the Support Disk. Read the provided disk for the most current programs.

A software disk is supplied with the initial purchase of a Unidex 14 controller to aid the user in developing their application program. It is supplied on a PC compatible floppy disk in both executable and source form. Install the U14C in your computer, install the disk in drive A: and type A:U14_POLL. The program will ask for a file name. Press enter and the program will announce it is in "Interactive mode". You are then communicating with the U14C board and can execute moves by typing them at the keyboard. Enter the following commands:

EN WY

This will turn on the echo from Unidex 14 so you can see what you are typing and should respond with the current firmware revision number so that you know that communication is established with the U14C board.

You are welcome to use the subroutines in this software for your application. No license is required. These programs are usable with all the PC/XT/AT compatible motion control products.

NOTE: **Software support Doc. — fault routine provides fault indication from all servo axis (not available on steppers) thru a limit and output line, which also indicates a cw or ccw limit has been encountered, Unidex 14's status register may be polled to test for a limit condition, which would invalidate the fault input, unless it occurred simultaneously with a limit.**

The following routines were written specifically for programming the Unidex 14. These routines perform the same function in both BASIC and C languages.

Home -	Sends all axes to the Home Limit Switch and then moves it out to the first Marker.
Init Sys -	Resets Unidex 14 and sets defaults
In Pos -	Waits until all drives are in position
Send File -	Reads a file containing commands and then sends them to the Unidex 14
Write Chr -	Sends a character to the Unidex 14, if there is room in the buffer, if there is no room, will wait
Check Status -	Checks to see if the Unidex 14 is flagging an error or has executed a id
Fault -	Returns true "1" if a fault condition exists

Write String -	Checks to see if the Unidex 14 has room for a character, if not, waits until there is room and then sends the character to Unidex 14
In Bit -	Returns the status of the user inputs
Remote -	Controls the Remote Clock and Direction inputs to the Unidex 14 Power Chassis (L is PC control, R is Clock and Direction)
Read Chr -	Waits for a character from Unidex 14
Read Data -	Checks to see if Unidex 14 has a character to send, if the Unidex 14 does it is returned
Read LF -	Reads characters until a CR LF is received then begins to input data until a CR LF is received
Read String -	Wait and input a specified number of characters from the Unidex 14

SECTION 9-2: README.DOC

Below is a description of the programs found on this disk.

U14_BAS. BAS	46508	--- Microsoft QuickBasic Version 4.50
JOG. C	29939	--- Microsoft C 5.1 prog. for cursor control of motors
U14_POLL. C	11960	--- MS QuickC 2.5 program using polling method
U14_INTR. C	16443	--- Microsoft C 5.1 program using interrupts
U14_CDMA.EXE	33430	--- DOS executable version of U14_CDMA.C
U14_BAS. EXE	80988	--- DOS executable version of U14_BAS.BAS
U14_PAS. PAS	4934	--- Microsoft QuickPascal
U14_POLL.EXE	27763	--- DOS executable version of U14_POLL.C
U14_CDMA. C	17418	--- Microsoft C Version 5.1 for DMA example
README. DOC	3531	--- This file
JOG. EXE	59778	--- DOS executable version of JOG.C

SECTION 9-3: BASIC DEMONSTRATION PROGRAM

```

DECLARE FUNCTION Limit$ (cmd!)
DECLARE FUNCTION Valid$ (char$)
DECLARE SUB Filter (outletter%)
DECLARE SUB LoadYCal ()
DECLARE SUB LoadZCal ()
DECLARE SUB LoadTCal ()
DECLARE FUNCTION Trans.Y$ (mode$, NewPos!)
DECLARE FUNCTION Trans.Z$ (mode$, NewPos!)
DECLARE FUNCTION Trans.T$ (mode$, NewPos!)
DECLARE FUNCTION Trans.X$ (mode$, NewPos!)
DECLARE SUB LoadXCal ()
DECLARE SUB Home ()           ' Unidex 14 Ver. 1.12
DECLARE SUB InitSys ()        ' 9/9/91
DECLARE SUB InPos (move$)
DECLARE SUB GetPosition ()
DECLARE SUB SendFile (filename$)
DECLARE SUB WriteChr (outletter%)
DECLARE SUB CheckStatus ()
DECLARE FUNCTION Fault% ()
DECLARE SUB WriteString (cmd$)
DECLARE FUNCTION InBit! ()
DECLARE SUB Remote (mode$)
DECLARE FUNCTION Power16& (power%)
DECLARE FUNCTION ReadChr% ()
DECLARE FUNCTION ReadData% ()
DECLARE FUNCTION ReadLF$ ()
DECLARE FUNCTION ReadString$ (count%)

' define Unidex 14 register I/O addresses, & bits of status register.

CONST datareg = 768, donereg = 769, controlreg = 770, statusreg = 771
CONST donebit = 16, ibfbit = 32, tbebit = 64, irqbit = 128
CONST cmderrbit = 1, initbit = 2, encbit = 4, ovrbit = 8
CONST cr = 13, LF = 10          ' Carriage Return, Line Feed

DEFINT A-Z
COMMON SHARED status%, sendmode
COMMON SHARED fileerror ' this will be set to a 1 in "handler -
    ' if the named file cannot be opened
COMMON SHARED AbsPosX AS LONG, AbsPosY AS LONG, AbsPosZ AS LONG, AbsPosT AS LONG
COMMON SHARED LastAbsX AS SINGLE, LastAbsY AS LONG, LastAbsZ AS LONG, LastAbsT AS LONG
COMMON SHARED LastErrX AS INTEGER, LastErrY AS INTEGER, LastErrZ AS INTEGER, LastErrT
AS INTEGER

```

```

COMMON SHARED CalSizX AS LONG, CalSizY AS LONG, CalSizZ AS LONG, CalSizT AS LONG
COMMON SHARED XUsrU!, YUsrU!, ZUsrU!, TUsrU!      ' user units
COMMON SHARED Xratio&, Yratio&, Zratio&, Tratio&    ' encoder ratio
DIM SHARED XErr(1000) AS INTEGER, YErr(1000) AS INTEGER, ZErr(1000) AS INTEGER, TErr(1000) AS INTE-
GER
COMMON SHARED name$

XUsrU! = 1: YUsrU! = 1 ' reset all user units to 1
ZUsrU! = 1: TUsrU! = 1
Xratio& = 1: Yratio& = 1 ' reset encoder ratio to 1
Zratio& = 1: Tratio& = 1
fileerror = 0

ON ERROR GOTO handler

COLOR 14, 1: CLS
OUT controlreg, 0      ' disable Unidex 14 interrupts
InitSys                ' initialize system

INPUT " Name of a Unidex 14 command file to download ( for no file): ", name$
PRINT CHR$(13); " Terminal emulation mode active, press CTRL C to exit"
SendFile name$
keystroke = 0
sendmode = 1           ' display U14 output mode
WHILE (keystroke < > 3) ' 3 is a Control-C
    key$ = INKEY$
    IF key$ < > "" THEN ' if a key was pressed
        keystroke = ASC(key$)
        WriteChr (keystroke) ' send key value to Unidex 14
    END IF

    result = ReadData
    IF result > 0 THEN ' get Unidex 14's output if there is any
        IF (result < > 0) AND (result < > 3) AND (result < > 13) THEN
            ' don't print null's or ^C or carriage returns
            PRINT CHR$(result);
        END IF
    END IF
    CheckStatus
WEND
END

handler: 'this routine handles run time errors

```

```

SELECT CASE ERR
CASE 52 'no name given or illegal name'
    fileerror = 1
    RESUME NEXT
CASE 53
    PRINT "This program could not find file: "; name$
    fileerror = 1
    RESUME NEXT
CASE ELSE
    'else display error message and attempt to resume program
    PRINT "Error # "; ERR: RESUME NEXT
END SELECT ' end of handler routine

DEFSNG A-Z
SUB CheckStatus

' Check to see if Unidex 14 is flagging an error or has executed an id

IF (status% AND donebit) < > 0 THEN
    IF (status% AND cmderrbit) < > 0 THEN
        PRINT
        PRINT "Unidex 14 Status Register indicates a command error"

    ELSEIF (status% AND encbit) < > 0 THEN
        PRINT
        PRINT "Unidex 14 Status Register indicates an Encoder Slip error"

    ELSEIF (status% AND ovrbit) < > 0 THEN
        PRINT
        PRINT "Unidex 14 Status Register indicates a limit switch is closed"

    ELSE
        ' an ID has been encountered, print the done axis
        doneflag = INP(donereg)
        IF doneflag < > 0 THEN
            PRINT
            PRINT "Unidex 14 Status Register indicates a DONE flag has been set."
            PRINT "AXES DONE is/are: ";
            IF doneflag MOD 2 = 1 THEN PRINT "x ";
            doneflag = doneflag \ 2
            IF doneflag MOD 2 = 1 THEN PRINT "y ";
            doneflag = doneflag \ 2
            IF doneflag MOD 2 = 1 THEN PRINT "z ";
            doneflag = doneflag \ 2
            IF doneflag MOD 2 = 1 THEN PRINT "t ";
            PRINT

```

```

PRINT "Enter Cntrl-Y or IC command to clear Done Flags"
  END IF
END IF
status% = INP(statusreg)
WHILE ((status% AND tbebit) = 0) ' wait till Unidex 14 is ready
  status% = INP(statusreg)
WEND
OUT datareg, 25 ' send CTRL-Y to reset status register

END IF
END SUB ' checkstatus

DEFINT A-Z
FUNCTION Fault

' This function returns true "1" if a fault condition exists, on any servo
' axis, stepper drives do not have a fault output to Unidex 14.

inputs& = InBit ' read the inputs
IF ((inputs& AND 64) = 0) THEN
  Fault = 0 ' no fault condition
ELSE
  Fault = 1 ' fault condition exists
END IF

END FUNCTION

DEFSNG A-Z
SUB GetPosition

' This sub-program returns the absolute position from Unidex 14.

SHARED AbsPosX AS LONG, AbsPosY AS LONG, AbsPosZ AS LONG, AbsPosT AS LONG

WriteString ("AzRP") ' request z position from Unidex 14
AbsPosZ = VAL(ReadLF$) ' input z position

WriteString ("AtRP") ' request t position from Unidex 14
AbsPosT = VAL(ReadLF$) ' input t position

WriteString ("AyRP") ' request y position from Unidex 14
AbsPosY = VAL(ReadLF$) ' input y position

WriteString ("AxRP") ' request x position from Unidex 14
AbsPosX = VAL(ReadLF$) ' input x position

```

```

' Translate Unidex 14's absolute positions to real world positions.
IF CalSizX = 0 THEN GOTO gskp1      ' is axis cal. present ?
IF AbsPosX = 0 THEN GOTO gskp1      ' only neg. # calibrated
APos = 0 - AbsPosX                  ' make it positive
Ap! = APos / CalSizX                ' find index into table
FPos = Ap! - INT(Ap!)               ' find fractional cal. step
Ap! = INT(Ap!)
AbsPosX = AbsPosX + FIX(FPos * (XErr(Ap! + 1) - XErr(Ap!)) + XErr(Ap!) + .5)

```

```

gskp1: IF CalSizY = 0 THEN GOTO gskp2      ' is axis cal. present ?
IF AbsPosY = 0 THEN GOTO gskp2      ' only neg. # calibrated
APos = 0 - AbsPosY                  ' make it positive
Ap! = APos / CalSizY                ' find index into table
FPos = Ap! - INT(Ap!)               ' find fractional cal. step
Ap! = INT(Ap!)
AbsPosY = AbsPosY + FIX(FPos * (YErr(Ap! + 1) - YErr(Ap!)) + YErr(Ap!) + .5)

```

```

gskp2: IF CalSizZ = 0 THEN GOTO gskp3      ' is axis cal. present ?
IF AbsPosZ = 0 THEN GOTO gskp3      ' only neg. # calibrated
APos = 0 - AbsPosZ                  ' make it positive
Ap! = APos / CalSizZ                ' find index into table
FPos = Ap! - INT(Ap!)               ' find fractional cal. step
Ap! = INT(Ap!)
AbsPosZ = AbsPosZ + FIX(FPos * (ZErr(Ap! + 1) - ZErr(Ap!)) + ZErr(Ap!) + .5)

```

```

gskp3: IF CalSizT = 0 THEN GOTO gskp4      ' is axis cal. present ?
IF AbsPosT = 0 THEN GOTO gskp4      ' only neg. # calibrated
APos = 0 - AbsPosT                  ' make it positive
Ap! = APos / CalSizT                ' find index into table
FPos = Ap! - INT(Ap!)               ' find fractional cal. step
Ap! = INT(Ap!)
AbsPosT = AbsPosT + FIX(FPos * (TErr(Ap! + 1) - TErr(Ap!)) + TErr(Ap!) + .5)

```

```

gskp4:

```

```

END SUB

```

```

SUB Home

```

```

' This sub-program homes all the axis by the hardware homing circuit.

```

```

' the next two lines home to limit switch at pre-programmed velocity, (VL cmd)
cmd$ = "axlfhlhm;aylfhlhm;azlfhlhm;atlfhlhm;": GOSUB wrt
cmd$ = "axhhhr;ln;ayhhhr;ln;azhhhr;ln;athhhhr;ln;": GOSUB wrt
FOR X = 1 TO 3000: NEXT X

```

```

' the next line homes the axis to the marker (absolute reference)
cmd$ = "aasaBL13;WT9;BH13;": GOSUB wrt ' initiate hardware home cycle
PRINT CHR$(13); "Homing !"
cmd$ = "AX": GOSUB wrt
FOR X = 1 TO 3000: NEXT X          ' wait for home to end

cmd$ = "ATLP0;": GOSUB wrt        ' set t axis position to zero
LastErrT = 0: LastAbsT = 0        ' reset t cal. variables
cmd$ = "AZLP0;": GOSUB wrt        ' set z axis position to zero
LastErrZ = 0: LastAbsZ = 0        ' reset z cal. variables
cmd$ = "AYLP0;": GOSUB wrt        ' set y axis position to zero
LastErrY = 0: LastAbsY = 0        ' reset y cal. variables
cmd$ = "AXLP0;": GOSUB wrt        ' set x axis position to zero
LastErrX = 0: LastAbsX = 0        ' reset x cal. variables
EXIT SUB

wrt: FOR X% = 1 TO LEN(cmd$)      ' loop till all chr's are sent
    WHILE (INP(statusreg) AND tbebit = 0) ' wait till Unidex 14 is ready
    WEND
    OUT datareg, ASC(MID$(cmd$, X%, 1)) ' send an ASCII chr. from string
NEXT X%
RETURN

END SUB

FUNCTION InBit

' This function returns the status of the user inputs.

cmd$ = "AXBX": GOSUB wrti        ' ask for input bit states
result$ = ReadLF$                ' input the hex values
FOR X% = 0 TO 5                  ' next line pre-scales the hex 'A-F' values
IF (ASC(MID$(result$, X% + 1, 1)) >= 65) THEN MID$(result$, X% + 1, 1) = CHR$(65 - 7)
accum = accum + (Power16(5 - X%) * (ASC(MID$(result$, X% + 1, 1)) - 48)) ' calc. base 10 value
NEXT X%
InBit = accum                    ' return dec. value of inputs
EXIT FUNCTION

wrti: FOR X% = 1 TO LEN(cmd$)    ' loop till all chr's are sent
    WHILE (INP(statusreg) AND tbebit = 0) ' wait till Unidex 14 is ready
    WEND
    OUT datareg, ASC(MID$(cmd$, X%, 1)) ' send an ASCII chr. from string
NEXT X%
RETURN

```

END FUNCTION

SUB InitSys

SHARED cal AS INTEGER

' This sub-program resets Unidex 14 and sets all system defaults.

```

WriteString ("RS")          ' reset to known state
LoadXCal                   ' load X axis cal. table
LoadYCal                   ' load Y axis cal. table
LoadZCal                   ' load Z axis cal. table
LoadTCal                   ' load T axis cal. table
WriteString ("AXEF")        ' disable Unidex 14 echo
WHILE (INP(statusreg) AND 32) ' buffer not empty ?
  X = ReadData              ' clr. it !
WEND
Remote ("L")               ' disable rem. clk. & dir.
WriteString ("WY")
result$ = ReadLF$           ' get version #
PRINT : PRINT " Unidex 14 Version # is = "; result$: PRINT
REM  WriteString ("AAPA1,1,1,1;") ' high I. mode during motion
WriteString ("AXBH13;")     ' reset home cycle pulse

```

END SUB

DEFINT A-Z

SUB InPos (move\$)

' Wait for all servo axis & stepper drives with 1MR ramper bds to come to rest.
 ' A "GO" command is expected within Move\$!

```

WriteString (move$)        ' start the move
PRINT "Waiting for drives to come to rest"
nloop: inputs& = InBit      ' read the inputs
IF ((inputs& AND 128) = 0) THEN GOTO nloop ' wait for move to start
innloop: inputs& = InBit    ' read the inputs
IF ((inputs& AND 128) = 128) THEN GOTO innloop ' wait for move to end

```

END SUB

DEFSNG A-Z

FUNCTION Limit\$ (cmd)

' Limit a cmd. string to the proper number of digits.

```
temp$ = STR$(cmd)           ' make it a string
dec.pt = INSTR(cmd$, ".")   ' find the decimal point
IF dec.pt = 0 THEN
    Limit$ = RIGHT$(temp$, 10) ' no dec. pt. return all
ELSE
    Limit$ = LEFT$(temp$, 11)  ' return 11 char.s
END IF
```

END FUNCTION

DEFINT A-Z
SUB LoadTCal

' Load T axis calibration table into memory.

```
LastAbsT = 0                ' initialize to start = 0
name$ = "t_axis.cal"
OPEN name$ FOR INPUT AS #1   ' open t cal. table
WHILE (char$ < > CHR$(3))    ' ignore header before CTRL-C
    char$ = INPUT$(1, #1)    ' get a character from file
WEND
INPUT #1, ClrCrLf$           ' clear CR LF from file
INPUT #1, CalSizT            ' get calibrated step size
INPUT #1, ClrCrLf$           ' clear CR LF from file
number = 0                   ' home pos = 0
WHILE (NOT EOF(1))           ' load T error table till EOF
    INPUT #1, TErr(number)    ' input error
    INPUT #1, ClrCrLf$        ' clear CR LF from file
    number = number + 1       ' increment subscript
WEND
CLOSE #1                     ' close the file
```

END SUB

SUB LoadXCal

' Load X axis calibration table into memory.

```
LastAbsX = 0                ' initialize to start = 0
name$ = "x_axis.cal"
OPEN name$ FOR INPUT AS #1   ' open x cal. table
WHILE (char$ < > CHR$(3))    ' ignore header before CTRL-C
    char$ = INPUT$(1, #1)    ' get a character from file
WEND
INPUT #1, ClrCrLf$           ' clear CR LF from file
```

```

INPUT #1, CalSizX      ' get calibrated step size
INPUT #1, ClrCrLf$    ' clear CR LF from file
number = 0             ' home pos = 0
WHILE (NOT EOF(1))    ' load X error table till EOF
  INPUT #1, XErr(number) ' input error
  INPUT #1, ClrCrLf$    ' clear CR LF from file
  number = number + 1   ' increment subscript
WEND
CLOSE #1              ' close the file

END SUB

SUB LoadYCal

' Load Y axis calibration table into memory.

LastAbsY = 0          ' initialize to start = 0
name$ = "y_axis.cal"
OPEN name$ FOR INPUT AS #1 ' open y cal. table
WHILE (char$ < > CHR$(3)) ' ignore header before CTRL-C
  char$ = INPUT$(1, #1) ' get a character from file
WEND
INPUT #1, ClrCrLf$    ' clear CR LF from file
INPUT #1, CalSizY     ' get calibrated step size
INPUT #1, ClrCrLf$    ' clear CR LF from file
number = 0             ' home pos = 0
WHILE (NOT EOF(1))    ' load Y error table till EOF
  INPUT #1, YErr(number) ' input error
  INPUT #1, ClrCrLf$    ' clear CR LF from file
  number = number + 1   ' increment subscript
WEND
CLOSE #1              ' close the file

END SUB

SUB LoadZCal

' Load Z axis calibration table into memory.

LastAbsZ = 0          ' initialize to start = 0
name$ = "z_axis.cal"
OPEN name$ FOR INPUT AS #1 ' open z cal. table
WHILE (char$ < > CHR$(3)) ' ignore header before CTRL-C
  char$ = INPUT$(1, #1) ' get a character from file
WEND
INPUT #1, ClrCrLf$    ' clear CR LF from file

```

```

INPUT #1, CalSizZ      ' get calibrated step size
INPUT #1, ClrCrLf$     ' clear CR LF from file
number = 0             ' home pos = 0
WHILE (NOT EOF(1))     ' load Z error table till EOF
    INPUT #1, ZErr(number) ' input error
    INPUT #1, ClrCrLf$   ' clear CR LF from file
    number = number + 1  ' increment subscript
WEND
CLOSE #1              ' close the file

```

END SUB

FUNCTION Power16& (power)

' This is called by the InBit function only !
 ' This function returns 16 raised to the power of power.

```

p& = 1
FOR i% = 1 TO power
    p& = p& * 16
NEXT i%
Power16 = p&          ' return result

```

END FUNCTION

DEFSNG A-Z

FUNCTION ReadChr%

' Wait for a character from Unidex 14.

```

rc1: status% = INP(statusreg)
IF ((status% AND ibfbit) = 0) THEN GOTO rc1 ' read status register
X% = INP(datareg)                          ' get a chr.
IF X% = 0 THEN GOTO rc1                    ' if null, get another
ReadChr = X%                              ' return valid chr.

```

END FUNCTION ' ReadChr

FUNCTION ReadData%

' Check to see if Unidex 14 has a character to send, if it does, get it.

```

status% = INP(statusreg) ' read Unidex 14's status reg.
IF (status% AND ibfbit) < > 0 THEN ' if a chr. is available,
    ReadData = INP(datareg) ' return it,
ELSE ' else,

```

```

    ReadData = 0      ' return 0.
END IF

END FUNCTION ' readdata

FUNCTION ReadLF$

' This function reads chr's till a CR is received then begins inputting
' the data until a CR LF is received.

    WHILE (ReadChr < > cr)      ' read till leading CR
    WEND
lp2: temp = ReadChr      ' input a chr.
    IF (temp = LF) THEN GOTO done      ' end of string ?
    result$ = result$ + CHR$(temp)      ' no; build string
    GOTO lp2      ' wait for LF
done: temp = ReadChr      ' input trailing CR
    ReadLF$ = result$      ' return string

END FUNCTION

FUNCTION ReadString$ (count%)

' Wait and input count% number of chr's from Unidex 14.

    FOR X% = 1 TO count%      ' read till count% chr's
        result$ = result$ + CHR$(ReadChr)      ' build string
    NEXT X%
    ReadString$ = result$      ' return result$

END FUNCTION

DEFINT A-Z
SUB Remote (mode$)

' This sub-program controls the remote clock & direction inputs to the
' Unidex 14 Power chassis. (L is PC control, R is clock & dir.)

    IF UCASE$(mode$) = "R" THEN
        WriteString ("AXBL12;")      ' set remote mode
    ELSE
        WriteString ("AXBH12;")      ' set local mode
    END IF

END SUB

```

SUB SendFile (filename\$)

' This sub-pgm. reads a file containing commands and sends them to Unidex 14.

```

OPEN filename$ FOR INPUT AS #1
IF fileerror = 0 THEN
    sendmode = 1          ' display U14 output mode
    DO UNTIL EOF(1)
        temp$ = INPUT$(1, #1) ' get one character from file
        charval = ASC(temp$) ' convert character to ASCII
        WriteChr (charval) ' send the chr. to Unidex 14
        charval = ReadData ' get chr. from Unidex 14's buffer
        IF (charval < > 13 AND charval < > 0) THEN PRINT CHR$(charval); ' display it !
    LOOP
END IF
CLOSE #1
sendmode = 0          ' don't display U14 output

```

END SUB ' sendfile

DEFSNG A-Z

FUNCTION Trans.T\$(mode\$, NewPos)

' Translate Absolute or Incremental T position to corrected position through
 ' that axis lookup table within memory. Axes must be homed before initially
 ' calling this function.

```

IF CalSizT = 0 THEN          ' is calibration active ?
    temp$ = STR$(NewPos) ' return same # as string
    GOTO xtt
END IF
mode$ = UCASE$(mode$)          ' convert to upper case
Steps = NewPos * Tratio& * TUsrU! ' convert to steps
Orig = NewPos          ' hold original move
NewPos = 0 - Steps      ' make it positive

IF mode$ < > "A" THEN NewPos = LastAbsT + NewPos - LastErrT ' convert to new absolute
ErrEt: IF NewPos < 0 THEN          ' if abs. pos. 0 then
    Trans.T = MID$(STR$(Orig), 2) ' return same # as a string
    EXIT FUNCTION          ' exit
END IF

temp = NewPos / CalSizT          ' find index into error table
pt = INT(temp)          ' find int. pointer
IF pt > 1000 THEN NewPos = -1: PRINT "T axis move out of range !": GOTO ErrEt
FracStep = temp - pt          ' find fractional cal. step

```

```

' find corrected absolute position
NewErr = FracStep * (TErr(pt + 1) - TErr(pt)) + TErr(pt)
NewAbsT = NewPos + NewErr      ' find new abs. position
IF mode$ <> "A" THEN            ' if incremental mode then
  IF TUsrU! = 1 THEN
    temp$ = STR$(0 - FIX(NewAbsT - LastAbsT / (Tratio& * TUsrU!) + .5))' convert to incremental
  ELSE
    temp$ = Limit(0 - ((NewAbsT - LastAbsT) / (Tratio& * TUsrU!))) ' convert to incr. int.
  END IF
ELSE                             ' else if absolute mode then
  IF TUsrU! = 1 THEN
    temp$ = STR$(0 - FIX(NewAbsT / (Tratio& * TUsrU!) + .5))' NewAbsT is correction value
  ELSE
    temp$ = Limit(0 - (NewAbsT / (Tratio& * TUsrU!)))
  END IF
END IF
LastAbsT = NewAbsT              ' remember LastAbsT position
LastErrT = NewErr               ' remember last error
xtt: IF VAL(temp$) <> 0 THEN      ' do i need to remove leading space
  Trans.T = temp$               ' no, return whole string
ELSE                             ' else
  Trans.T = MID$(temp$, 2)      ' return without leading space
END IF

```

END FUNCTION

FUNCTION Trans.X\$ (mode\$, NewPos)

```

' Translate Absolute or Incremental X position to corrected position through
' that axis lookup table within memory. Axes must be homed before initially
' calling this function.

```

```

IF CalSizX = 0 THEN              ' is calibration active ?
  temp$ = STR$(NewPos)           ' no; return same # as string
  GOTO xtx
END IF
mode$ = UCASE$(mode$)           ' convert to upper case
Steps = NewPos * Xratio& * XUsrU! ' convert to steps
Orig = NewPos                    ' hold original move
NewPos = 0 - Steps               ' make it positive

IF mode$ <> "A" THEN NewPos = LastAbsX + NewPos - LastErrX ' convert to new absolute
ErrEx: IF NewPos < 0 THEN        ' if abs. pos. 0 then
  Trans.X = MID$(STR$(Orig), 2) ' return same # as a string
EXIT FUNCTION                    ' exit

```

END IF

```

temp = NewPos / CalSizX      ' find index into error table
px = INT(temp)              ' find int. pointer
IF px > 1000 THEN NewPos = -1: PRINT " X axis move out of range !": GOTO ErrEx
FracStep = temp - px        ' find fractional cal. step

' find corrected absolute position
NewErr = FracStep * (XErr(px + 1) - XErr(px)) + XErr(px)
NewAbsX = NewPos + NewErr    ' find new abs. position
IF mode$ <> "A" THEN          ' if incremental mode then
  IF XUsrU! = 1 THEN
    temp$ = STR$(0 - FIX(NewAbsX - LastAbsX / (Xratio& * XUsrU!) + .5))' convert to incr.
  ELSE
    temp$ = Limit(0 - ((NewAbsX - LastAbsX) / (Xratio& * XUsrU!))) ' convert to incr. int.
  END IF
ELSE                          ' else if absolute mode then
  IF XUsrU! = 1 THEN
    temp$ = STR$(0 - FIX(NewAbsX / (Xratio& * XUsrU!) + .5))' NewAbsX is correction value
  ELSE
    temp$ = Limit(0 - (NewAbsX / (Xratio& * XUsrU!)))' NewAbsX is correction value
  END IF
END IF
LastAbsX = NewAbsX          ' remember LastAbsX position
LastErrX = NewErr           ' remember last error
xtx: IF VAL(temp$) < 0 THEN   ' do i need to remove leading space
  Trans.X = temp$           ' no, return whole string
ELSE                         ' else
  Trans.X = MID$(temp$, 2)   ' return without leading space
END IF

```

END FUNCTION

FUNCTION Trans.Y\$ (mode\$, NewPos)

' Translate Absolute or Incremental Y position to corrected position through
 ' that axis lookup table within memory. Axes must be homed before initially
 ' calling this function.

```

IF CalSizY = 0 THEN          ' is calibration active ?
  temp$ = STR$(NewPos)       ' return same # as string
  GOTO xty
END IF
mode$ = UCASE$(mode$)        ' convert to upper case
Steps = NewPos * Yratio& * YUsrU! ' convert to steps
Orig = NewPos                ' hold original move

```

```

NewPos = 0 - Steps          ' make it positive

IF mode$ < > "A" THEN NewPos = LastAbsY + NewPos - LastErrY ' convert to new absolute
ErrEy: IF NewPos < 0 THEN          ' if abs. pos. < 0 then
    Trans.Y = MID$(STR$(Orig), 2) ' return same # as a string
    EXIT FUNCTION                ' exit
END IF

temp = NewPos / CalSizY      ' find index into error table
py = INT(temp)               ' find int. pointer
IF py > 1000 THEN NewPos = -1: PRINT "Y axis move out of range !": GOTO ErrEy
FracStep = temp - py        ' find fractional cal. step

' find corrected absolute position
NewErr = FracStep * (YErr(py + 1) - YErr(py)) + YErr(py)
NewAbsY = NewPos + NewErr    ' find new abs. position
IF mode$ < > "A" THEN        ' if incremental mode then
    IF YUsrU! = 1 THEN
        temp$ = STR$(0 - FIX(NewAbsY - LastAbsY / (Yratio& * YUsrU!) + .5)) ' convert to incremental
    ELSE
        temp$ = Limit(0 - ((NewAbsY - LastAbsY) / (Yratio& * YUsrU!))) ' convert to incr. int.
    END IF
ELSE                          ' else if absolute mode then
    IF YUsrU! = 1 THEN
        temp$ = STR$(0 - FIX(NewAbsY / (Yratio& * YUsrU!) + .5)) ' NewAbsY is correction value
    ELSE
        temp$ = Limit(0 - (NewAbsY / (Yratio& * YUsrU!)))
    END IF
END IF

LastAbsY = NewAbsY          ' remember LastAbsY position
LastErrY = NewErr           ' remember last error
xty: IF VAL(temp$) < 0 THEN  ' do i need to remove leading space
    Trans.Y = temp$         ' no, return whole string
ELSE                          ' else
    Trans.Y = MID$(temp$, 2) ' return without leading space
END IF

END FUNCTION

FUNCTION Trans.Z$(mode$, NewPos)

' Translate Absolute or Incremental Z position to corrected position through
' that axis lookup table within memory. Axes must be homed before initially
' calling this function.

IF CalSizZ = 0 THEN          ' is calibration active ?

```

```

temp$ = STR$(NewPos)      ' return same # as string
GOTO xtz
END IF
mode$ = UCASE$(mode$)      ' convert to upper case
Steps = NewPos * Zratio& * ZUsrU!  ' convert to steps
Orig = NewPos              ' hold original move
NewPos = 0 - Steps        ' make it positive

IF mode$ <> "A" THEN NewPos = LastAbsZ + NewPos - LastErrZ ' convert to new absolute
ErrEz: IF NewPos < 0 THEN
    Trans.Z = MID$(STR$(NewPos), 2) ' return same # as string
    EXIT FUNCTION                  ' exit
END IF

temp = NewPos / CalSizZ      ' find index into error table
pz = INT(temp)              ' find int. pointer
IF pz > 1000 THEN NewPos = -1: PRINT "Z axis move out of range!": GOTO ErrEz
FracStep = temp - pz        ' find fractional cal. step

' find corrected absolute position
NewErr = FracStep * (ZErr(pz + 1) - ZErr(pz)) + ZErr(pz)
NewAbsZ = NewPos + NewErr    ' find new abs. position
IF mode$ <> "A" THEN          ' if incremental mode then
    IF ZUsrU! = 1 THEN
        temp$ = STR$(0 - FIX(NewAbsZ - LastAbsZ / (Zratio& * ZUsrU!) + .5)) ' convert to incremental
    ELSE
        temp$ = Limit(0 - ((NewAbsZ - LastAbsZ) / (Zratio& * ZUsrU!))) ' convert to incr. int.
    END IF
ELSE                          ' else if absolute mode then
    IF ZUsrU! = 1 THEN
        temp$ = STR$(0 - FIX(NewAbsZ / (Zratio& * ZUsrU!) + .5)) ' NewAbsZ is correction value
    ELSE
        temp$ = Limit(0 - (NewAbsZ / (Zratio& * ZUsrU!)))
    END IF
END IF
LastAbsZ = NewAbsZ          ' remember LastAbsZ position
LastErrZ = NewErr           ' remember last error
xtz: IF VAL(temp$) < 0 THEN   ' do i need to remove leading space
    Trans.Z = temp$          ' no, return whole string
ELSE                          ' else
    Trans.Z = MID$(temp$, 2) ' return without leading space
END IF

END FUNCTION

FUNCTION Valid$ (char$)

```

```
' This sub - program returns the character it receives if it is valid,
' else it returns null.
```

```
char$ = UCASE$(char$)      ' make upper case character
SELECT CASE (char$)
CASE (" ")
char$ = ""      ' space is not valid
CASE (CHR$(13))
char$ = ""      ' CR is not valid
CASE (CHR$(10))
char$ = ""      ' LF is not valid
CASE (",")
char$ = ""      ' ; is not valid
END SELECT
Valid = char$      ' return character
```

```
END FUNCTION
```

```
DEFINT A-Z
```

```
SUB WriteChr (outletter%) STATIC
```

```
' Send a chr. to Unidex 14 if there is room in its buffer; else wait !
```

```
' if axis calibration is not desired, un-comment the next line !
```

```
' goto sema
```

```
IF mode$ = "" THEN mode$ = "X"      ' AX mode default
```

```
IF echo THEN      ' should i display it ?
```

```
IF (outletter% 0) AND (outletter% 3) AND (outletter% 10) THEN
```

```
' don't print null's or ^ C or Carriage return's
```

```
PRINT CHR$(outletter%);
```

```
END IF
```

```
END IF
```

```
IF term THEN
```

```
IF ((outletter% = 13) OR (outletter% = 10) OR (outletter% = 32) OR (outletter = 59)) THEN term = 0
```

```
GOTO sema
```

```
END IF
```

```
IF sema.colon THEN      ' am i waiting for a sema colon ?
```

```
IF (outletter% = 59) THEN sema.colon = 0 ' got it, exit !
```

```
GOTO sema
```

```
END IF
```

```
IF param THEN GOTO bld.prm      ' build parameters
```

```
char$ = char$ + Valid(CHR$(outletter%)) ' build valid 2 char. cmd$
```

```
IF LEN(char$) 2 THEN GOTO sema ELSE GOTO here ' wait for 2 char. cmd$
```

```

bld.prm: IF mode$ = "A" THEN      ' look for proper mode terminator
  IF (outletter% = 59) THEN GOTO bexit ' if "," goto bexit
  ELSE
    IF ((outletter% = 13) OR (outletter% = 10) OR (outletter% = 32) OR (outletter = 59)) THEN GOTO bexit ' terminate on CR,LF,SPC,or sema
  END IF
  IF (outletter% = 44) THEN pnum = pnum + 1: GOTO wexit ' if "," goto wexit
  prm$(pnum) = prm$(pnum) + CHR$(outletter%) ' build parameter number pnum
  GOTO wexit

aa.at: SELECT CASE (mode$)      ' translate & output positions
  CASE ("A")
    prm$(0) = Trans.X(md$, VAL(prm$(0)))
    prm$(1) = Trans.Y(md$, VAL(prm$(1)))
    prm$(2) = Trans.Z(md$, VAL(prm$(2)))
    prm$(3) = Trans.T(md$, VAL(prm$(3)))
  CASE ("M")
    prm$(0) = Trans.X(md$, VAL(prm$(0)))
    prm$(1) = Trans.Y(md$, VAL(prm$(1)))
    prm$(2) = Trans.Z(md$, VAL(prm$(2)))
    prm$(3) = Trans.T(md$, VAL(prm$(3)))
  CASE ("X")
    prm$(0) = Trans.X(md$, VAL(prm$(0)))
  CASE ("Y")
    prm$(1) = Trans.Y(md$, VAL(prm$(1)))
  CASE ("Z")
    prm$(2) = Trans.Z(md$, VAL(prm$(2)))
  CASE ("T")
    prm$(3) = Trans.T(md$, VAL(prm$(3)))
  END SELECT

writeit: FOR X = 0 TO pnum      ' write each parameter
  FOR y = 1 TO LEN(prm$(X))    ' write each chr. of param.
    WHILE (INP(statusreg) AND tbebit = 0) ' wait till Unidex 14 is ready
    WEND
    OUT datareg, ASC(MID$(prm$(X), y, 1)) ' send character to Unidex 14
  NEXT y
  IF (X = pnum) THEN
    WHILE (INP(statusreg) AND tbebit = 0) ' wait till Unidex 14 is ready
    WEND
    OUT datareg, ASC(",") ' send "," to Unidex 14
  END IF
NEXT X
  WHILE (INP(statusreg) AND tbebit = 0) ' wait till Unidex 14 is ready
  WEND
  OUT datareg, ASC(";") ' send terminator to Unidex 14

```

CheckStatus

RETURN

bexit: do.it = 1 ' set execution flag

here: SELECT CASE (char\$)

CASE ("AA")

mode\$ = "A" ' all axis mode

CASE ("AC")

IF mode\$ = "A" THEN

sema.colon = 1 ' ignore all till ";" received

ELSE

term = 1 ' wait for terminator

END IF

CASE ("AF")

IF mode\$ = "A" THEN

sema.colon = 1 ' ignore all till ";" received

ELSE

term = 1 ' wait for terminator

END IF

CASE ("AM")

mode\$ = "M" ' multitasking axis mode

CASE ("AN")

IF mode\$ = "A" THEN

sema.colon = 1 ' ignore all till ";" received

ELSE

term = 1 ' wait for terminator

END IF

CASE ("AT")

mode\$ = "T" ' T axis mode

CASE ("AX")

mode\$ = "X" ' X axis mode

CASE ("AY")

mode\$ = "Y" ' Y axis mode

CASE ("AZ")

mode\$ = "Z" ' Z axis mode

CASE ("BH")

term = 1 ' ignore all till terminator

CASE ("BL")

term = 1 ' ignore all till terminator

CASE ("CD")

IF (do.it = 0) THEN

param = 4

ELSE

temp.mode\$ = mode\$ ' remember mode

md\$ = "A" ' set abs. mode

```

mode$ = "X"      ' fake AX mode
GOSUB aa.at      ' translate & output positions
mode$ = temp.mode$ ' restore mode
' remember contour plane definition
p1 = 1: ax$ = "" ' reset pointer & string
IF LEN(prm$(0)) > 0 THEN MID$(ax$, p1, 1) = ax$ + "X": p1 = p1 + 1 ' set X axis
IF LEN(prm$(1)) > 0 THEN MID$(ax$, p1, 1) = ax$ + "Y": p1 = p1 + 1 ' set Y axis
IF LEN(prm$(2)) > 0 THEN MID$(ax$, p1, 1) = ax$ + "Z": p1 = p1 + 1 ' set Z axis
IF LEN(prm$(3)) > 0 THEN MID$(ax$, p1, 1) = ax$ + "T": p1 = p1 + 1 ' set T axis
ax$ = MID$(ax$, 1, 2) ' limit to 2 axis
END IF
CASE ("CR")
IF (do.it = 0) THEN
param = 3
ELSE ' translate param. 1&2 thru proper cal. tables
SELECT CASE (ax$)
CASE ("XY")
prm$(0) = Trans.X(md$, VAL(prm$(0)))
prm$(1) = Trans.Y(md$, VAL(prm$(1)))
CASE ("XZ")
prm$(0) = Trans.X(md$, VAL(prm$(0)))
prm$(1) = Trans.Z(md$, VAL(prm$(1)))
CASE ("XT")
prm$(0) = Trans.X(md$, VAL(prm$(0)))
prm$(1) = Trans.T(md$, VAL(prm$(1)))
CASE ("YZ")
prm$(0) = Trans.Y(md$, VAL(prm$(0)))
prm$(0) = Trans.Z(md$, VAL(prm$(1)))
CASE ("YT")
prm$(0) = Trans.Y(md$, VAL(prm$(0)))
prm$(1) = Trans.T(md$, VAL(prm$(1)))
CASE ("ZT")
prm$(0) = Trans.Z(md$, VAL(prm$(0)))
prm$(1) = Trans.T(md$, VAL(prm$(1)))
END SELECT
GOSUB writeit ' output corrected parameters
END IF
CASE ("CV")
term = 1 ' ignore all till terminator
CASE ("EF")
echo = 0 ' kill echo
CASE ("EN")
outletter% = 70 ' change it to E'F'
echo = 1 ' but; enable my echo
CASE ("ER") ' enc. counts, motor counts
IF (do.it = 0) THEN

```

```

    param = 2
ELSE
    SELECT CASE (mode$)
        CASE ("X")
            Xratio& = INT(VAL(prm$(1)) / VAL(prm$(0)))
        CASE ("Y")
            Yratio& = INT(VAL(prm$(1)) / VAL(prm$(0)))
        CASE ("Z")
            Zratio& = INT(VAL(prm$(1)) / VAL(prm$(0)))
        CASE ("T")
            Tratio& = INT(VAL(prm$(1)) / VAL(prm$(0)))
    END SELECT
    GOSUB writeit
END IF
CASE ("ES")
    term = 1      ' ignore all till terminator
CASE ("FP")
    IF (do.it = 0) THEN
        param = 1
    ELSE
        md$ = "A"    ' set abs. mode
        GOSUB aa.at  ' translate & output positions
    END IF
CASE ("HD")
    term = 1      ' wait for terminator
CASE ("HG")
    term = 1      ' ignore all till terminator
CASE ("HM")
    IF (do.it = 0) THEN
        outletter% = 72    ' change it to H'H"
        param = 1
    ELSE
        Home
        GOSUB writeit
    END IF
CASE ("HR")
    IF (do.it = 0) THEN
        outletter% = 72    ' change it to H'H"
        param = 1
    ELSE
        Home
        GOSUB writeit
    END IF
CASE ("HV")
    term = 1      ' ignore all till terminator
CASE ("IN")

```

```

IF mode$ = "A" THEN
    sema.colon = 1 ' ignore all till ";" received
ELSE
    term = 1      ' wait for terminator
END IF
CASE ("JG")
    term = 1      ' ignore all till terminator
CASE ("KM")
    Home
CASE ("KR")
    Home
CASE ("LP")      ' load position register
    IF (do.it = 0) THEN
        param = 1
    ELSE
        GOSUB writeit      ' send to Unidex 14
    END IF
CASE ("LS")
    term = 1      ' ignore all till terminator
CASE ("MA")
    IF (do.it = 0) THEN
        IF mode$ = "A" THEN param = 4 ELSE param = 1
    ELSE
        md$ = "A"      ' set abs. mode
        GOSUB aa.at    ' translate & output positions
    END IF
CASE ("ML")
    IF (do.it = 0) THEN
        IF mode$ = "A" THEN param = 4 ELSE param = 1
    ELSE
        md$ = "I"      ' set incremental mode
        GOSUB aa.at    ' translate & output positions
    END IF
CASE ("MR")
    IF (do.it = 0) THEN
        IF mode$ = "A" THEN param = 4 ELSE param = 1
    ELSE
        md$ = "I"      ' set mode
        GOSUB aa.at    ' go translate inc. positions
    END IF
CASE ("MT")
    IF (do.it = 0) THEN
        IF mode$ = "A" THEN param = 4 ELSE param = 1
    ELSE
        md$ = "A"      ' set absolute mode
        GOSUB aa.at    ' translate & output positions

```

```

END IF
CASE ("MV")      ' move vel. (abs. pos., velocity)
  IF (do.it = 0) THEN
    param = 2
  ELSE
    md$ = "A"    ' set abs. mode
    GOSUB aa.at   ' translate abs. positions
  END IF
CASE ("PA")
  IF mode$ = "A" THEN
    sema.colon = 1 ' ignore all till ";" received
  ELSE
    term = 1      ' wait for terminator
  END IF
CASE ("PN")
  IF mode$ = "A" THEN
    sema.colon = 1 ' ignore all till ";" received
  ELSE
    term = 1      ' wait for terminator
  END IF
CASE ("RE")
  IF (mode$ <> "A") AND (mode$ <> "M") THEN
    IF sendmode THEN
      WHILE (INP(statusreg) AND tbebit = 0)
        WEND          ' wait till U14 is rdy
      OUT datareg, outletter% ' send "E" to U14
      CheckStatus
      result$ = ReadLF$    ' clr misc. encoder position
      EncPos& = VAL(result$) ' get encoder pos.

' translate Unidex 14's absolute positions to real world positions.

      IF EncPos& = > 0 THEN GOTO skp.cal    ' only neg. # calibrated
      SELECT CASE (mode$)
        CASE ("X")
          IF CalSizX = 0 THEN GOTO skp.cal
          IF EncPos& = > 0 THEN GOTO skp.cal ' only neg. # calibrated
          APos! = 0 - EncPos&          ' make it positive
          Ap! = APos! / CalSizX        ' find index into table
          FPos! = Ap! - INT(Ap!)      ' find fractional cal. step
          Ap! = INT(Ap!)
          EncPos& = EncPos& + FIX(FPos! * (XErr(Ap! + 1) - XErr(Ap!)) + XErr(Ap!) + .5)
        CASE ("Y")
          IF CalSizY = 0 THEN GOTO skp.cal
          IF EncPos& = > 0 THEN GOTO skp.cal ' only neg. # calibrated
          APos! = 0 - EncPos&          ' make it positive

```

```

    Ap! = APos! / CalSizY      ' find index into table
    FPos! = Ap! - INT(Ap!)    ' find fractional cal. step
    Ap! = INT(Ap!)
    EncPos& = EncPos& + FIX(FPos! * (YErr(Ap! + 1) - YErr(Ap!)) + YErr(Ap!) + .5)
CASE ("Z")
    IF CalSizZ = 0 THEN GOTO skip.cal
    IF EncPos& >= 0 THEN GOTO skip.cal ' only neg. # calibrated
    APos! = 0 - EncPos&        ' make it positive
    Ap! = APos! / CalSizZ      ' find index into table
    FPos! = Ap! - INT(Ap!)    ' find fractional cal. step
    Ap! = INT(Ap!)
    EncPos& = EncPos& + FIX(FPos! * (ZErr(Ap! + 1) - ZErr(Ap!)) + ZErr(Ap!) + .5)
CASE ("T")
    IF CalSizT = 0 THEN GOTO skip.cal
    IF EncPos& >= 0 THEN GOTO skip.cal ' only neg. # calibrated
    APos! = 0 - EncPos&        ' make it positive
    Ap! = APos! / CalSizT      ' find index into table
    FPos! = Ap! - INT(Ap!)    ' find fractional cal. step
    Ap! = INT(Ap!)
    EncPos& = EncPos& + FIX(FPos! * (TErr(Ap! + 1) - TErr(Ap!)) + TErr(Ap!) + .5)
END SELECT
skip.cal:      PRINT CHR$(13); EncPos&
    char$ = ""    ' clr cmd$
    GOTO wexit
END IF
END IF
CASE ("RM")
    term = 1      ' ignore all till terminator
CASE ("RP")
    IF sendmode THEN
        WHILE (INP(statusreg) AND tbebit = 0)
        WEND      ' wait till U14 is rdy
        OUT datareg, outletter% ' send "P" to U14
        CheckStatus
        result$ = ReadLF$    ' clr misc. position
        prm$(0) = "AA;RP"    ' define pos. cmd. string
        pnum = 0: GOSUB writeit ' write it to U14
        prm$(0) = ""        ' clr before exit
        result$ = ReadLF$    ' input all 4 position's
        AbsPosX = VAL(result$) ' get x abs. pos.
        LastComma = INSTR(result$, ",") + 1 ' find start of y pos.
        AbsPosY = VAL(MID$(result$, LastComma, 25)) ' get y abs. pos.
        LastComma = INSTR(LastComma, result$, ",") + 1 ' find start of z pos.
        AbsPosZ = VAL(MID$(result$, LastComma, 25)) ' get z abs. pos.
        LastComma = INSTR(LastComma, result$, ",") + 1 ' find start of t pos.
        AbsPosT = VAL(MID$(result$, LastComma, 25)) ' get t abs. pos.

```

' translate Unidex 14's absolute positions to real world positions.

IF CalSizX = 0 THEN GOTO skp1 ' is axis cal. present ?

IF AbsPosX > = 0 THEN GOTO skp1 ' only neg. # calibrated

APos! = 0 - AbsPosX ' make it positive steps

Ap! = APos! / CalSizX ' find index into table

FPos! = Ap! - INT(Ap!) ' find fractional cal. step

Ap! = INT(Ap!)

AbsPosX = AbsPosX + FIX(FPos! * (XErr(Ap! + 1) - XErr(Ap!)) + XErr(Ap!) + .5)

skp1: IF CalSizY = 0 THEN GOTO skp2 ' is axis cal. present ?

IF AbsPosY > = 0 THEN GOTO skp2 ' only neg. # calibrated

APos! = 0 - AbsPosY ' make it positive

Ap! = APos! / CalSizY ' find index into table

FPos! = Ap! - INT(Ap!) ' find fractional cal. step

Ap! = INT(Ap!)

AbsPosY = AbsPosY + FIX(FPos! * (YErr(Ap! + 1) - YErr(Ap!)) + YErr(Ap!) + .5)

skp2: IF CalSizZ = 0 THEN GOTO skp3 ' is axis cal. present ?

IF AbsPosZ > = 0 THEN GOTO skp3 ' only neg. # calibrated

APos! = 0 - AbsPosZ ' make it positive

Ap! = APos! / CalSizZ ' find index into table

FPos! = Ap! - INT(Ap!) ' find fractional cal. step

Ap! = INT(Ap!)

AbsPosZ = AbsPosZ + FIX(FPos! * (ZErr(Ap! + 1) - ZErr(Ap!)) + ZErr(Ap!) + .5)

skp3: IF CalSizT = 0 THEN GOTO skp4 ' is axis cal. present ?

IF AbsPosT > = 0 THEN GOTO skp4 ' only neg. # calibrated

APos! = 0 - AbsPosT ' make it positive

Ap! = APos! / CalSizT ' find index into table

FPos! = Ap! - INT(Ap!) ' find fractional cal. step

Ap! = INT(Ap!)

AbsPosT = AbsPosT + FIX(FPos! * (TErr(Ap! + 1) - TErr(Ap!)) + TErr(Ap!) + .5)

skp4: SELECT CASE (mode\$)

CASE ("A")

PRINT CHR\$(13); AbsPosX; ","; AbsPosY; ","; AbsPosZ; ","; AbsPosT

CASE ("M")

PRINT CHR\$(13); AbsPosX; ","; AbsPosY; ","; AbsPosZ; ","; AbsPosT

CASE ("X")

PRINT CHR\$(13); AbsPosX

CASE ("Y")

PRINT CHR\$(13); AbsPosY

CASE ("Z")

PRINT CHR\$(13); AbsPosZ

CASE ("T")

```

PRINT CHR$(13); AbsPosT
END SELECT

WHILE (INP(statusreg) AND tbebit = 0)
WEND      ' wait till U14 is rdy
OUT datareg, 65      ' send "A" to U14
WHILE (INP(statusreg) AND tbebit = 0)
WEND      ' wait till U14 is rdy
OUT datareg, ASC(mode$) ' send char. to reset mode
CheckStatus
char$ = ""      ' clr cmd$
GOTO wexit
END IF
CASE ("RS")
mode$ = "X"      ' reset to AX mode
XUsrU! = 1: YUsrU! = 1 ' reset all user units to 1
ZUsrU! = 1: TUsrU! = 1
Xratio& = 1: Yratio& = 1 ' reset encoder ratio to 1
Zratio& = 1: Tratio& = 1
echo = 0      ' kill echo
LastAbsX = 0: LastAbsY = 0 ' reset last axis cal. pos.
LastAbsZ = 0: LastAbsT = 0
LastErrX = 0: LastErrY = 0 ' reset last cal. error
LastErrZ = 0: LastErrT = 0
CASE ("SP")
IF (do.it = 0) THEN
param = 1
ELSE
md$ = "A"      ' set abs. mode
GOSUB aa.at      ' translate positions
END IF
CASE ("SW")
term = 1      ' ignore all till terminator
CASE ("UF")      ' set user units off
SELECT CASE (mode$)      ' kill user units
CASE ("X")
XUsrU! = 1      ' user units = 1
CASE ("Y")
YUsrU! = 1      ' user units = 1
CASE ("Z")
ZUsrU! = 1      ' user units = 1
CASE ("T")
TUsrU! = 1      ' user units = 1
END SELECT
CASE ("UU")
IF (do.it = 0) THEN

```

```

    param = 1
ELSE
    SELECT CASE (mode$) ' store user units
        CASE ("X")
            XUsrU! = VAL(prm$(0))
        CASE ("Y")
            YUsrU! = VAL(prm$(0))
        CASE ("Z")
            ZUsrU! = VAL(prm$(0))
        CASE ("T")
            TUsrU! = VAL(prm$(0))
    END SELECT
    GOSUB writeit
END IF
CASE ("VB")
    IF mode$ = "A" THEN
        sema.colon = 1 ' ignore all till ";" received
    ELSE
        term = 1 ' wait for terminator
    END IF
CASE ("VL")
    IF mode$ = "A" THEN
        sema.colon = 1 ' ignore all till ";" received
    ELSE
        term = 1 ' wait for terminator
    END IF
CASE ("VS")
    term = 1 ' wait for terminator
CASE ("WS")
    term = 1 ' wait for terminator
CASE ("WT")
    term = 1 ' wait for terminator
END SELECT
IF do.it THEN
    pnum = 0 ' clear param. pointer
    param = 0 ' clear # of param's
    do.it = 0 ' i did it, so clear it
    prm$(0) = "": prm$(1) = "" ' clear param. storage
    prm$(2) = "": prm$(3) = ""
END IF
IF (param = 0) THEN char$ = "" ' clear cmd$

sema: status% = INP(statusreg)
WHILE ((status% AND tbebit) = 0) ' wait till Unix 14 is ready
    status% = INP(statusreg)
WEND

```

```
OUT datareg, outletter%      ' send character to Unidex 14

wexit:

END SUB ' writechr

DEFSNG A-Z
SUB WriteString (cmd$)

' Check if Unidex 14 has room for a chr., if it doesn't, wait until there is
'   room, then send the chr. to Unidex 14.

FOR X% = 1 TO LEN(cmd$)      ' loop till all chr's are sent
  WriteChr (ASC(MID$(cmd$, X%, 1))) ' send an ASCII chr. from string
  CheckStatus
NEXT X%

END SUB ' writestring
```

SECTION 9-4: PASCAL DEMONSTRATION PROGRAM

PROGRAM U14_PAS;

{This program allows the user to interact with a Unidex 14 motion control board. It uses a "polling" method to communicate with the board as opposed to using interrupts as the "C" version does.

Portions of this program may be adapted by Unidex 14 users for application programs. No license is required.

This program was written using Microsoft QuickPascal Version 1.0 Copyright 1989
Version 1.00
RELEASED July 4, 1990 }

USES
 CRT;

CONST
 DATA_REG = \$300;
 DONE_REG = \$301;
 CONTROL_REG = \$302;
 STATUS_REG = \$303;

```

CMDERR_BIT = $01;
INIT_BIT = $02;
ENC_BIT = $04;
OVRT_BIT = $08;
DONE_BIT = $10;
IBF_BIT = $20;
TBE_BIT = $40;
IRQ_BIT = $80;

```

```

VAR

```

```

    keystroke, u14dataout, status : byte;
    name : string;

```

```

PROCEDURE outp( outaddress : word; outdata : byte);
{output a byte to the given I/O address}

```

```

BEGIN

```

```

    INLINE

```

```

    (
        $8B/$96/outaddress/ { mov dx,outaddress ;Load I/O Address into DX reg.}
        $8A/$86/outdata/   { mov al,outdata   ;Load Data byte into AL reg.}
        $EE                { out dx,al       ;Output byte in AL to DX addr}
    );

```

```

END; {procedure outp}

```

```

FUNCTION inp(inaddress: word) : byte;
{input a byte from given I/O address}

```

```

VAR

```

```

    return : byte;

```

```

BEGIN

```

```

    INLINE

```

```

    (
        $8B/$96/inaddress/ { mov dx, inaddress ;Load I/O Address into DX reg}
        $EC/               { in  al, dx       ;Get byte at DX add, put it in AL}
        $88/$86/return     { mov return, a    ;Put AL byte in Return variable}
    );

```

```

    inp := return;

```

```

END; {function inp}

```

```

PROCEDURE write_u14_data(outletter : byte);

```

{check if Unidex 14 has room for a byte, if it doesn't wait until there is room, then send the byte to it }

```
BEGIN
  WHILE inp(STATUS_REG) AND TBE_BIT = 0 DO
    BEGIN
      END;
    outp(DATA_REG,outletter);
  END; {procedure write_u14_data}
```

```
PROCEDURE checkstatus (status : byte);
{check to see if Unidex 14 is flagging an error or has executed an id}
```

```
VAR
  doneflag : byte;
BEGIN
  IF (status AND DONE_BIT) < > 0 THEN
    BEGIN
      IF (status AND CMDERR_BIT) < > 0 THEN
        BEGIN
          writeln;
          writeln ('Unidex 14 Status Register indicates a command error');
        END
      ELSE IF (status AND ENC_BIT) < > 0 THEN
        BEGIN
          writeln;
          writeln ('Unidex 14 Status Register indicates an Encoder Slip error');
        END
      ELSE IF (status AND OVRT_BIT) < > 0 THEN
        BEGIN
          writeln;
          writeln ('Unidex 14 Status Register indicates limit switch closed');
        END
      ELSE {an ID has been encountered, print the done axis}
        BEGIN
          doneflag := INP(DONE_REG);
          IF doneflag < > 0 THEN
            BEGIN
              writeln;
              writeln ('Unidex 14 Status Register indicates a DONE flag set');
              write ('AXES DONE is/are: ');
              IF doneflag MOD 2 = 1 THEN
```

```

        write('x ');
        doneflag := doneflag DIV 2;
        IF doneflag MOD 2 = 1 THEN
            write('y ');
            doneflag := doneflag DIV 2;
            IF doneflag MOD 2 = 1 THEN
                write('z ');
                doneflag := doneflag DIV 2;
                IF doneflag MOD 2 = 1 THEN
                    write('t ');
                END;
            END;
        END;
        write_u14_data(24); {clear error status}
    END;
END; {procedure checkstatus}

FUNCTION read_u14_data : byte ;

BEGIN
    {check to see if Unix 14 has a character to send, if it does, print it}
    status := inp(STATUS_REG);
    IF (status AND IBF_BIT) 0 THEN
        read_u14_data := inp(DATA_REG)
    ELSE
        read_u14_data := 0;
    END;
END; {procedure read_u14_data}

PROCEDURE sendfile(filename:string);
    {this procedure reads a file containing Unix 14 commands and send them to
    Unix 14}

VAR
    c : char;
    filepointer : FILE OF char;

    {SI-} {turn off run time I/O error checking}

BEGIN
    assign(filepointer , filename);
    reset(filepointer);
    IF (IOresult < > 0) THEN
        BEGIN

```

```

    writeln('Unable to open file: ',filename);
    exit;
END;
WHILE NOT eof(filepointer) DO
    BEGIN
        read(filepointer,c);
        write_u14_data( ord (c) );
    END;

{$I+} {turn on run time I/O error checking}

    close(filepointer);
END; {procedure sendfile}

BEGIN {MAIN PROGRAM}

    outp(CONTROL_REG,$00); {disable Unidex 14's interrupts}
    clrscr;
    write('Name of a Unidex 14 command file to download (<CR> for no file):');
    readln(name);
    sendfile(name);
    writeln(output,'Interactive mode, Enter Unidex 14 commands, Press ^C to exit');
    keystroke := 0;
    WHILE keystroke <> 3 DO { 3 is a Control-C}
        BEGIN
            IF keypressed THEN
                BEGIN
                    keystroke := Ord(readkey);
                    write_u14_data(keystroke);
                    IF keystroke = 13 THEN { if it's a CR print an LF too}
                        write( chr(10));
                    END;

                    u14dataout := read_u14_data; {get Unidex 14 output }
                    IF (u14dataout 0) AND (u14dataout 3) THEN
                        {don't print null's or ^C}
                        write( chr(u14dataout));

                    checkstatus(status);
                END;
            END;
        END.

```

SECTION 9-5: C DEMONSTRATION PROGRAM WITH INTERRUPTS

```
/* PROGRAM U14_INTR.C          Version 1.01
```

This program allows the user to interact with the Unidex 14 motion control board. This program uses the PC's interrupt line IRQ5 to communicate between the PC and Unidex 14. All C functions changing the mode of Unidex 14 will return it to the AX mode.

This program will have to be restarted if an RS (reset) command is sent to Unidex 14.

Portions of this program may be adapted by users for application programs using Unidex 14. No license is required.

This program was written using Microsoft C 5.1 Compiler.

RELEASED Aug. 27, 1991

```
*/
```

```
#include <stdio.h>
#include <dos.h>
#include <conio .h>
```

```
#define DATA_REG    0x300 /* Unidex 14 I/O port definitions */
#define DONE_REG     0x301
#define CONTROL_REG   0x302
#define STATUS_REG    0x303
```

```
#define CMDERR_BIT 0x1 /* CONTROL and STATUS register bit definitions */
#define INIT_BIT   0x2
#define ENC_BIT    0x4
#define OVRT_BIT   0x8
#define DONE_BIT   0x10
#define IBF_BIT    0x20
#define TBE_BIT    0x40
#define IRQ_BIT    0x80
```

```
/* PC interrupt definitions */
```

```
#define ICNT    0x20 /* interrupt terminator byte for 8259 */
#define IMREG   0x21 /* interrupt mask register I/O address */
#define ILEVEL  0x20 /* interrupt mask level 5 byte */
```

```
#define CNTRL_C 3 /* character definitions */
#define CLEAR_D 24 /* Control-X */
```

```
#define LF      10
#define CR      13

#define BSIZE   1000      /* input and output buffer size */

int  inxti, inxt0, onxti, onxt0; /* buffer pointers */

char done_flag, error_flag;
char inbuf[ BSIZE ], outbuf[ BSIZE ]; /* define buffers arrays */

char fname[80];           /* file name buffer */

long atol(char *);        /* function prototypes */
char toupper(char);
void main(void);
void sendfile(void);
void interact(void);
void checkstatus(void);
char getcm(void);
void init_sys(void);
void putcm(char c);
void writestring(char *cmd);
int  mhit(void);
interrupt far service(void);
void readstring(int cnt, char *result);
char readchr(void);
int  fault(void);
void getposition(long *);
void home(void);
long inbit(void);
void inpos(char move[]);
long power_16(int power);
void readlf(char *result);
void remote(char mode);
void system(char *);
int  strlen(char *);
char *strtok(char *,char *);

void main(void)
{
    long lpoint[4];          /* pointer to all 4 axis pos. */
    inxti = inxt0 = onxti = onxt0 = 0; /* set buffer pointers to beginning */
    done_flag = 0;           /* clear done flag variable */
    error_flag = 0;          /* clear error flag variable */

    init(service);           /* initialize interrupts */
```

```

system("cls");          /* clear PC's screen */
init_sys();             /* set some defaults */
sendfile();             /* send file to Unidex 14 */
interact();             /* interactive mode */
printf("\nInputs = %li", inbit()); /* display input states */
if(fault())             /* test for drive faults */
    printf("\nERROR drive fault exists!");
printf("\nDrives are now In Position");
getposition(lpoin);     /* update the absolute position's */
printf("\n X axis position = %li", lpoin[0]);
printf("\n Y axis position = %li", lpoin[1]);
printf("\n Z axis position = %li", lpoin[2]);
printf("\n T axis position = %li", lpoin[3]);
/*   home();           /* send all axis home */
}

void sendfile(void) /* Open file and send it to Unidex 14 */
{
    FILE *fp;

    printf("\nName of Unidex 14 command file to download (< CR > for no file):");
    gets(fname);
    if(strlen(fname) > 0)
    {
        if ((fp = fopen(fname, "r")) == NULL)
            printf("\nCan't open %s\n", fname);
        else
        {
            while(!feof(fp) == 0) /* while not end of file */
            {
                putcm(fgetc(fp)); /* get a char from file, send it to Unidex 14 */
                /* band aid - if DOS turns interrupt off, turn it back on */
                outp(IMREG, inp(IMREG) & ~ ILEVEL);
                while(mhit()) /* while a char available from Unidex 14 */
                {
                    putch(getcm()); /* print it */
                }
                checkstatus();
            }
            fclose(fp); /* close file when done */
            printf("%s transmitted.\n", fname);
        }
    }
}

void interact(void) /* Sends keyboard entry to Unidex 14, and displays Unidex 14's output */
{
    char outchar = 0, inchar;

    printf("\nInteractive mode, Enter Unidex 14 commands, Press Control-C to exit.\n");

```

```

while (outchar != CNTRL_C)
{
    if (kbhit()) /* char available from keyboard ? */
    {
        outchar = getch(); /* get char from keyboard */
        putcm(outchar); /* send it to Unidex 14 */
        if (outchar == CR)
            putchar(LF); /* add linefeeds when return is pressed to
                           avoid overstriking letters on the screen */
    }
    if (mhit()) /* char available from Unidex 14 ? */
    {
        inchar = getcm(); /* get char */
        putchar(inchar); /* display it */
    }
    checkstatus();
}
}

void checkstatus(void) /* this routine decodes Unidex 14's status and displays it's info */
{
    extern char done_flag, error_flag;

    if (error_flag != 0)
    {
        if ((error_flag & CMDERR_BIT) != 0) /* command error */
            printf("\nUnidex 14's Status Register indicates a command error\n");
        else
            if ((error_flag & ENC_BIT) != 0) /* encoder slip */
                printf("\nUnidex 14's Status Register indicates an Encoder Slip error\n");
            else
                if ((error_flag & OVRT_BIT) != 0) /* limit switch is active */
                    printf("\nUnidex 14's Status Register indicates a limit switch is closed\n");
                error_flag = 0; /* clear error_flag */
    }

    if (done_flag != 0) /* an ID has been encountered, print the done axis */
    {
        printf("\nUnidex 14's Status Register indicates a DONE flag has been set\n");
        printf("AXES DONE: ");
        if (done_flag % 2 == 1)
            printf("x ");
        done_flag = done_flag / 2;
        if (done_flag % 2 == 1)
            printf("y ");
        done_flag = done_flag / 2;
        if (done_flag % 2 == 1)
            printf("z ");
        done_flag = done_flag / 2;
        if (done_flag % 2 == 1)
            printf("t ");
    }
}

```

```

printf("\nPress Control-Y, or type IC to clear Unidex 14's done status.\n");
done_flag = 0; /* clear done flag */
}
}

char getcm(void) /* Returns character from Unidex 14 */
{
char c;
if(inxti == inxto) /* no char available in buffer */
return (0);
else
{
c = inbuff[inxto + +]; /* get char. and incr. pointer */
if(inxto == BSIZE) /*if end of buffer */
inxto = 0; /* reset pointer */
outp(CONTROL_REG, inp(CONTROL_REG) | IBF_BIT); /* enable U14 intr. */
return (c);
}
}

void putcm(char c) /* Put character into Unidex 14's buffer */
{
int i, j;

i = onxti; /* set local pointer to global input pointer of output buffer*/
j = i + 1;
if(j == BSIZE) /* if next pointer points to the end, wrap pointer to start */
j = 0;
while(j == onxto) /* while there is no room in the buffer for more */
; /* wait until Unidex 14's interrupt routine makes room in buffer */
outbuff[i + +] = c; /* get character i points to and increment i */
if(i == BSIZE) /* if i points to the end of buffer wrap it around */
i = 0;
onxti = i; /* set global pointer to local pointer's value */
outp(CONTROL_REG, inp(CONTROL_REG) | TBE_BIT); /* enable Unidex 14 to interrupt */
}

mhit() /* returns non zero if input buffer has character */
{
return(inxto - inxti);
}

interrupt far service() /* interrupt driven routine to communicate with Unidex 14 */
{
char c;
int i;

```

```

c = inp(STATUS_REG) & (inp(CONTROL_REG) | 0x0f);
/* get the status of interrupts that are enabled */

if(c & DONE_BIT) /* if the DONE bit is true */
{
    done_flag |= inp(DONE_REG); /* save done axis flags */
    error_flag |= (c & 0x0f); /* save error bits also */
    outp(DATA_REG, CLEARD); /* reset Unidex 14 status register DONE bit
        without clearing flags shown in RA or
        RI commands which ^Y does */
}
else
if((c & TBE_BIT) == TBE_BIT) /* if Unidex 14 will accept a char */
{
    if(onxto != onxti) /* do we have a char to send to Unidex 14 ? */
    {
        outp(DATA_REG, outbuf[onxto]); /* send char to Unidex 14 */
        if(++onxto == BSIZE) /* take care of wrap around */
            onxto = 0;
        if(onxto == onxti) /* if the buffer is now empty */
            /* turn off Unidex 14 interrupt so it won't keep asking for chars */
            outp(CONTROL_REG, inp(CONTROL_REG) & ~TBE_BIT);
    }
    else /* no char in buffer to send to Unidex 14 */
        /* turn off Unidex 14's interrupt so it won't keep asking for chars */
        outp(CONTROL_REG, inp(CONTROL_REG) & ~TBE_BIT);
}
if((c & IBF_BIT) == IBF_BIT) /* if Unidex 14 has a char to send to the PC */
{
    inbuf[inxti] = inp(DATA_REG); /* get the char and put it in the buffer */
    if(++inxti == BSIZE) /* take care of wrap around */
        inxti = 0;
    if((i = inxti + 1) == BSIZE) /* room for another char in the buffer? */
        i = 0;
    if(i == inxto)
        outp(CONTROL_REG, inp(CONTROL_REG) & ~IBF_BIT);
    /* no room in buffer, turn off interrupt so Unidex 14 cannot send more */
}
outp(ICNT, 0x20); /* send 8259 chip a return from interrupt */

/* PC/XT/AT interrupt control is edge sensitive only.
The following generates a rising edge if a second interrupt has occurred
prior to the first being reset. This can happen if more than one
source is enabled on the Unidex 14 as in this program. */

c = inp(CONTROL_REG);
outp(CONTROL_REG, c & ~IRQ_BIT); /* turn off interrupts */
outp(CONTROL_REG, c); /* and back on to create edge */
}

```

```

init(intvec) /* connect hardware interrupt IRQ5 to interrupt handler */
int (*intvec)();
{
    extern int  intdosx();
    union REGS inregs, outregs;
    struct SREGS segregs;

    inregs.h.ah = 0x25;      /* set interrupt vector DOS call */
    inregs.h.al = 13;        /* interrupt vector level 5 */
    segregs.ds = FP_SEG(*intvec); /* send the vector */
    inregs.x.dx = FP_OFF(*intvec);
    intdosx(&inregs, &outregs, &segregs);
    outp(IMREG, inp(IMREG) & ~ ILEVEL); /* enable interrupts */
    outp(CONTROL_REG, IBF_BIT | DONE_BIT | IRQ_BIT); /* turn on Unidex 14's interrupts */
}

void home(void) /* this function will hardware home all axis */
{
    long result;
    writestring("BL13;WT9;BH13;"); /* send all axis home */
    printf("\nHoming");
    writestring("AX"); /* set x axis active */
    for(result=0; result < 100; result++) /* wait for home to start */
    {
        result = inbit(); /* read the inputs */
        while ((result & 32) != 0) /* wait for home completion */
            result = inbit(); /* read the inputs */
        writestring("ATLP0;"); /* set t axis pos. 0 */
        writestring("AZLP0;"); /* set z axis pos. 0 */
        writestring("AYLP0;"); /* set y axis pos. 0 */
        writestring("AXLP0;"); /* set x axis pos. 0 */
        printf("\nAt Home");
    }

    long inbit(void) /* this function returns the state of the inputs */
    {
        int x;
        long accum = 0; /* decimal equiv. accumulator */
        char result[20]; /* init. string */
        writestring("AXBX"); /* ask for input bit states */
        readlf(result); /* input the hex-ascii values */
        for(x=0; x < 5; x++)
        {
            if (result[x] == 'A') /* is it 'A' thru 'F' ? */
                result[x] = result[x]-7; /* pre-scale hex values */
            accum = accum + (power_16(5-x) * (result[x]-48)); /* calc. base ten value */
        }
    }
}

```

```

    }
    return(accum);          /* return inputs in dec. */
}

long power_16(int power) /* this function returns 16 raised to the power of
    power. This is called by the inbit function only ! */
{
    int i;
    long p = 1;
    for(i=1; i <= power; i++)
        p = p * 16;
    return(p);
}

int fault(void) /* this function returns true '1' if any drive fault exists,
    on any servo axis, stepper drives do not have a fault
    output to Unidex 14. */
{
    if((inbit() & 64) == 0) /* read the inputs */
        return(0);        /* drive fault not present */
    else
        return(1);        /* drive fault present */
}

void inpos(char move[255]) /* this function starts a move, then waits for all
    servos, & steppers with 1MR bds. to come to rest.
    the move cmd string is expected to include a "go" cmd */
{
    writestring(move);      /* start the move */
    printf("\nWaiting for drives to come to rest");
    while(!(inbit() & 128)) /* wait for move to start */
        ;
    while(inbit() & 128)    /* wait for move to end */
        ;
}

void remote(char mode) /* this function controls the remote clk. & dir. inputs */
    /* to the Unidex 14 power chassis. (L is PC control, */
    /* R is remote clock & direction */
{
    if((toupper(mode) == 'R')) /* convert mode chr. to upper case */
        writestring("AXBL12;"); /* set Remote mode */
    else
        writestring("AXBH12;"); /* set Local mode */
}

void getposition(long *position) /* this function returns a pointer to 4 long int's */

```

```

{
char result[25];
    writestring("ATRP");      /* request position from Unidex 14 */
    readlf(result);
    position[3] = atol(result); /* find the t axis position */
    writestring("AZRP");      /* request position from Unidex 14 */
    readlf(result);
    position[2] = atol(result); /* find the z axis position */
    writestring("AYRP");      /* request position from Unidex 14 */
    readlf(result);
    position[1] = atol(result); /* find the y axis position */
    writestring("AXRP");      /* request position from Unidex 14 */
    readlf(result);
    position[0] = atol(result); /* find the x axis position */
}

void init_sys(void) /* this function initializes Unidex 14 system */
{
    int c;
    char result[40];
    writestring("AXEF");      /* kill echo if active */
    for (c=0; c < 3000; c++); /* delay till echo stops */
    {;}
    while ((inp(STATUS_REG) & IBF_BIT)) /* while there's a chr. in buffer */
        c = getcm(); /* input it */
    remote('L'); /* disable remote clk. & dir. */
    writestring("WY"); /* get Unidex 14 version number */
    readlf(result); /* input version number */
    printf("Version # is %s\n",result);
    /* writestring("AAPA1,1,1,1,"); /* set high i mode during moves */
    writestring("AXBH13,"); /* reset home cycle pulse */
}

void writestring(char cmd[255]) /* this function writes a string to Unidex 14 */
{
    int x = 0; /* set initial value */
    putcm('\x0d'); /* write a leading CR */
    while (cmd[x] != '\0') /* loop till all chr's sent */
    {
        putcm(cmd[x++]); /* write a chr. to Unidex 14 */
        checkstatus(); /* check Unidex 14's status register */
    }
}

char readchr(void) /* wait for a valid chr. and then return it */
{

```

```

char chr = 0;
while(chr == NULL)      /* read till valid chr. */
    chr = getcm();      /* get a chr. from Unidex 14 */
return(chr);
}

void readstring(int cnt, char *result) /* read cnt number of chr's and modify pointer to string */
{
int x;
for(x=0; x=(cnt-1); x++);      /* read cnt number of chr's */
{    result[x] = readchr();      /* get a valid chr. */
}
result[x++] = '\0';            /* add NULL terminator */
}

void readlf(char *result) /* read chr's till a LF is received then begin inputting string until CR LF */
{
int temp = 0;

while(readchr() != CR)        /* read till leading CR */
    ;
while((result[temp] = readchr()) != '\n') /* read till LF terminator */
{    temp++;                    /* build chr. string */
}
result[temp] = '\0';          /* add terminator */
while(readchr() != '\r')      /* read till LF terminator */
    ;
}

```

SECTION 9-6: U14_POLL.C

/* PROGRAM U14_POLL.C Version 1.01

This program allows the user to interact with the Unidex 14 motion control board. This program uses polling to communicate between the PC and Unidex 14. All C functions changing the mode of Unidex 14 will return it to the AX mode.

Portions of this program may be adapted by users for application programs using Unidex 14. No license is required.

This program was written using Microsoft QuickC 2.5 Compiler.

RELEASED Aug. 27, 1991

```
*/

#include <stdio.h>
#include <dos.h>
#include <conio.h>

#define DATA_REG    0x300 /* Unidex 14 I/O port definitions */
#define DONE_REG     0x301
#define CONTROL_REG  0x302
#define STATUS_REG   0x303

#define CMDERR_BIT 0x1 /* CONTROL and STATUS register bit definitions */
#define INIT_BIT   0x2
#define ENC_BIT    0x4
#define OVRT_BIT   0x8
#define DONE_BIT   0x10
#define IBF_BIT    0x20
#define TBE_BIT    0x40
#define IRQ_BIT    0x80

#define CNTRL_C 3 /* character definitions */
#define CLEARD 24 /* Control-X */
#define LF 10
#define CR 13

int status;
char fname[80]; /* file name buffer */

long atol(char *); /* function prototypes */
char toupper(char);
void main(void);
void sendfile(void);
void interact(void);
void checkstatus(void);
char getcm(void);
void init_sys(void);
void putcm(char c);
void writestring(char *cmd);
void getposition(long *position);
void readlf(char *result);
void writechr(int c);
char readchr(void);
int fault(void);
void home(void);
```

```

long inbit(void);
void inpos(char move[]);
long power_16(int power);
void remote(char mode);
void system(char *);
int strlen(char *);
char *strtok(char *, char *);

void main(void)
{
long lpoint[4];           /* array of all four axis pos's */

    system("cls");        /* clear PC's screen */
    init_sys();           /* set some defaults */
    /* home();            /* send all axis home */
    sendfile();           /* send file to Unidex 14 */
    interact();           /* interactive mode */
    printf("\nInputs = %li",inbit()); /* display input states */
    if(fault())           /* test for drive faults */
        printf("\nERROR drive fault exists !");
    getposition(lpoint);   /* update the absolute position's */
    printf("\n X axis position = %li",lpoint[0]);
    printf("\n Y axis position = %li",lpoint[1]);
    printf("\n Z axis position = %li",lpoint[2]);
    printf("\n T axis position = %li",lpoint[3]);
}

void sendfile(void) /* Open file and send it to Unidex 14 */
{
    FILE *fp;
    char x;

    printf("\nName of Unidex 14 command file to download (<CR> for no file):");
    gets(fname);
    if(strlen(fname) 0)
    {
        if ((fp = fopen(fname, "r")) == NULL)
            printf("\nCan't open %s", fname);
        else
        {
            while(!feof(fp) == 0) /* while not end of file */
            {

                putcm(fgetc(fp)) ; /* get a char from file, send it to Unidex 14 */
                while((x = getcm()) != NULL) /* while a char available from Unidex 14 */
                {

```

```

        putchar(x);    /* print it */
    }
    checkstatus();
}
fclose(fp);    /* close file when done */
printf("%s transmitted.\n", fname);
}
}
}

void interact(void) /* Sends keyboard entry to Unidex 14, and displays Unidex 14's output */
{
    char outchar = 0;
    int inchar;

    printf("\nInteractive mode, Enter Unidex 14 commands, Press Control-C to exit.\n");
    while (outchar != CNTRL_C)
    {
        if (kbhit())    /* char available from keyboard ? */
        {
            outchar = getch();    /* get char from keyboard */
            putcm(outchar);    /* send it to Unidex 14 */
            if(outchar == CR)
                putchar(LF);    /* add linefeeds when return is pressed to
                                avoid overstriking letters on the screen */
        }
        if ((inchar = getcm()) != NULL)    /* char available from Unidex 14 ? */
        {
            putchar(inchar);    /* display it */
        }

        checkstatus();
    }
}

void checkstatus(void) /* this routine decodes Unidex 14's status and displays it's info */
{
    extern int status;
    int done_flag;
    if ((status & DONE_BIT) != 0)
    {
        if ((status & CMDERR_BIT) != 0)    /* command error */
            printf("\nUnidex 14's Status Register indicates a command error\n");
        else
            if ((status & ENC_BIT) != 0)    /* encoder slip */
                printf("\nUnidex 14's Status Register indicates an Encoder Slip error\n");
        else

```

```

if ((status & OVRT_BIT) != 0)    /* limit switch is active */
    printf("\nUnidex 14's Status Register indicates a limit switch is closed\n");
else
{
    done_flag = inp(DONE_REG);
    if(done_flag != 0)           /* if an axis finished, display it */
    {
        printf("\nUnidex 14's Status Register indicates a DONE flag has been set\n");
        printf("AXES DONE: ");
        if(done_flag % 2 == 1)
            printf("x ");
        done_flag = done_flag / 2;
        if(done_flag % 2 == 1)
            printf("y ");
        done_flag = done_flag / 2;
        if(done_flag % 2 == 1)
            printf("z ");
        done_flag = done_flag / 2;
        if(done_flag % 2 == 1)
            printf("t ");
        printf("\n");
        done_flag = 0;
    }
}
putc(24); /* clr. error status */
}

char getcm(void) /* Returns character from Unidex 14 */
{
extern int status;
    status = inp(STATUS_REG); /* get U14's status */
    if((inp(STATUS_REG) & IBF_BIT)) /* if char available in buffer */
        return (inp(DATA_REG)); /* return it, */
    else /* else */
        return (0); /* return 0 */
}

void putcm(char c) /* Put character into Unidex 14's buffer */
{
    while((inp(STATUS_REG) & TBE_BIT) == 0) /* wait till Unidex 14's ready */
        ;
    outp(DATA_REG,c); /* write the chr. */
}

void home(void) /* this function will hardware home all axis */
{

```

```

long result;
writestring("BL13;WT9;BH13;");    /* send all axis home */
printf("\nHoming");
writestring("AX");                /* set x axis active */
for(result=0; result < 100; result++) /* wait for home to start */
{
    result = inbit();              /* read the inputs */
    while ((result & 32) != 0)      /* wait for home completion */
        result = inbit();         /* read the inputs */
    writestring("ATLP0;");          /* set t axis pos. 0 */
    writestring("AZLP0;");          /* set z axis pos. 0 */
    writestring("AYLP0;");          /* set y axis pos. 0 */
    writestring("AXLP0;");          /* set x axis pos. 0 */
    printf("\nAt Home");
}

long inbit(void) /* this function returns the state of the inputs */
{
    int x;
    long accum = 0;                /* decimal equiv. accumulator */
    char result[20];               /* init. string */
    writestring("AXBX");           /* ask for input bit states */
    readlf(result);                /* input the hex-ascii values */
    for(x=0; x < 5; x++)
    {
        if (result[x] == 'A')      /* is it 'A' thru 'F' ? */
            result[x] = result[x]-7; /* pre-scale hex values */
        accum = accum + (power_16(5-x) * (result[x]-48)); /* calc. base ten value */
    }
    return(accum);                /* return inputs in dec. */
}

long power_16(int power) /* this function returns 16 raised to the power of
    power. This is called by the inbit function only ! */
{
    int i;
    long p = 1;
    for(i=1; i < power; i++)
        p = p * 16;
    return(p);
}

int fault(void) /* this function returns true '1' if any drive fault exists,
    on any servo axis, stepper drives do not have a fault
    output to Unidex 14. */
{
    if((inbit() & 64) == 0) /* read the inputs */

```

```

    return(0);      /* drive fault not present */
else
    return(1);      /* drive fault present */
}

void inpos(char move[255]) /* this function starts a move, then waits for all
    servos, & steppers with 1MR bds. to come to rest.
    the move cmd string is expected to include a "go" cmd */
{
    writestring(move);      /* start the move */
    printf("\nWaiting for drives to come to rest");
    while(!(inbit() & 128)) /* wait for move to start */
        ;
    while(inbit() & 128)     /* wait for move to end */
        ;
}

void remote(char mode) /* this function controls the remote clk. & dir. inputs */
    /* to the Unidex 14 power chassis. (L is PC control, */
    /* R is remote clock & direction */
{
    if((toupper(mode) == 'R')) /* convert mode chr. to upper case */
        writestring("AXBL12;"); /* set Remote mode */
    else
        writestring("AXBH12;"); /* set Local mode */
}

void getposition(long *position) /* this function returns a pointer to 4 long int's */
{
    char result[25];
    writestring("ATRP"); /* request position from Unidex 14 */
    readlf(result);
    position[3] = atol(result); /* find the t axis position */
    writestring("AZRP"); /* request position from Unidex 14 */
    readlf(result);
    position[2] = atol(result); /* find the z axis position */
    writestring("AYRP"); /* request position from Unidex 14 */
    readlf(result);
    position[1] = atol(result); /* find the y axis position */
    writestring("AXRP"); /* request position from Unidex 14 */
    readlf(result);
    position[0] = atol(result); /* find the x axis position */
}

void init_sys(void) /* this function initializes Unidex 14 system */
{
    int c;

```

```

char result[40];
writestring("AXEF");      /* kill echo if active */
for (c=0; c < 767; c + +); /* delay till echo stops */
{; }
while ((inp(STATUS_REG) & IBF_BIT)) /* while there's a chr. in buffer */
    c = getcm(); /* input it */
remote('L'); /* disable remote clk. & dir. */
writestring("WY"); /* get Unidex 14 version number */
readlf(result); /* input version number */
printf("Version # is %s\n",result);
/* writestring("AA PA1,1,1,1;"); /* set high i mode during moves */
writestring("AXBH13;"); /* reset home cycle pulse */
}

void writestring(char cmd[255]) /* this function writes a string to Unidex 14 */
{
    int x = 0; /* set initial value */
    putcm('\x0d'); /* write a leading CR */
    while (cmd[x] != '\0') /* loop till all chr's sent */
    {
        putcm(cmd[x + +]); /* write a chr. to Unidex 14 */
        checkstatus(); /* check Unidex 14's status register */
    }
}

char readchr(void) /* wait for a valid chr. and then return it */
{
    extern int status;
    char chr = 0;
    status = inp(STATUS_REG); /* get U14's status */
    while(chr == NULL) /* read till valid chr. */
        chr = getcm(); /* get a chr. from Unidex 14 */
    return(chr);
}

void readstring(int cnt, char *result) /* read cnt number of chr's and modify pointer to string */
{
    int x;

    for(x=0; x == (cnt-1); x + +); /* read cnt number of chr's */
    {
        result[x] = readchr(); /* get a valid chr. */
    }
    result[x + +] = '\0'; /* add NULL terminator */
}

void readlf(char *result) /* read chr's till a LF is received then begin inputting string until CR LF */
{

```

```

int temp = 0;

while(readchr() != CR)      /* read till leading CR */
;
while((result[temp] = readchr()) != '\n') /* read till LF terminator */
{
    temp++;                /* build chr. string */
}
result[temp] = '\0';        /* add terminator */
while(readchr() != '\r')    /* read till LF terminator */
;
}

```

SECTION 9-7: C DEMONSTRATION PROGRAM WITH DMA

/* PROGRAM U14_CDMA.C Version 1.01

This program allows the user to interact with Unidex 14.

It includes a routine to test the DMA link from the host computer to Unidex 14. This requires that a jumper be placed on the DRQ1 and DACK1 pins of JP26 on Unidex 14.

Portions of this program may be adapted by Unidex 14 users for application programs. No license is required.

This program was written using Microsoft C 5.1 Optimizing Compiler.

This version released Aug. 27, 1991

```

*/

#include <stdio.h>
#include <dos.h>
#include <direct.h>

#define DATA_REG 0x300 /* Unidex 14 I/O port definitions */
#define DONE_REG 0x301
#define CONTROL_REG 0x302
#define STATUS_REG 0x303

#define CMDERR_BIT 0x1 /* CONTROL and STATUS register bit definitions */
#define ENC_BIT 0x4
#define OVRT_BIT 0x8
#define DONE_BIT 0x10
#define IBF_BIT 0x20

```

```

#define TBE_BIT 0x40
#define IRQ_BIT 0x80

#define DMASTAT 0x8    /* DMA definitions */
#define MASKREG 0x0a
#define DMAMODE 0x0b   /* dma mode register */
#define BPFF 0x0c
#define ADRREG 0x2     /* address register channel 1 */
#define COUNT 0x3      /* byte counter register channel 1 */
#define PGREG 0x83     /* page register channel 1 */
#define DMAEN 0x2
#define DMARD 0x1

#define ICNT 0x20       /* interrupt terminator for 8259 */
#define IMREG 0x21      /* interrupt mask register */
#define ILEVEL 0x20     /* interrupt mask level 5 */

#define CNTRLC 3        /* character definitions */
#define CLEAR 0x18      /* Control-X */
#define LF 10
#define CR 13

#define BSIZE 1000     /* buffer size */

void checkstatus(void); /* function prototypes */
long atol(char *);
char toupper(char);
void system(char *);
int strlen(char *);
char *strtok(char *, char *);
void home(void);
long inbit(void);
long power_16(int power);
int fault(void);
void inpos(char move[]);
void remote(char mode);
void init_sys(void);
void writestring(char cmd[]);
void getposition(long *position);
char readchr(void);
void readlf(char *result);
void sendfile(void);
interrupt far service();
void dma(int count, char *buffer);
void initdma(int count, long address);
void interact(void);

```

```

char getcm(void);
void putcm(char c);
int mhit(void);

int inxti, inxto, onxti, onxto; /* buffer pointers */
char inbuf[BSIZE], outbuf[BSIZE]; /* buffers */
char done_flag, error_flag;
char fname[80];
FILE *fp, *fopen();

/* sample command string that is sent to Unidex 14 via DMA */

#define MSG "en wy rp mr100000 go wy rp"
char buffer[] = MSG;
int count = sizeof(buffer);

void main(void)
{
long lpoint[4];
    system("cls"); /* clear the screen */
    printf("\nThis program includes a routine to test the DMA link from the host\
computer\nto Unidex 14. This requires that a jumper be placed on the DRQ1\
and DACK1\npins of JP26 on Unidex 14. It does not use DMA to interact with\
the board.\nIt uses interrupts and I/O ports, the same as the program U14_INTR.C does.\n");
    inxti = inxto = onxti = onxto = 0; /* buffers empty */
    init(service); /* init interrupts */
    init_sys(); /* set some defaults */
    dma(count, buffer); /* send sample string to Unidex 14 board via DMA */
/*    home(); /* send all axis home */
    interact(); /* go into interactive mode */
    printf("\nInput's = %li", inbit()); /* display input states */
    if(fault())
        printf("\nERROR drive fault exists !");
    printf("Drives are now In Position");
    getposition(lpoint); /* update the absolute positions */
    printf("\n X axis position = %li", lpoint[0]);
    printf("\n Y axis position = %li", lpoint[1]);
    printf("\n Z axis position = %li", lpoint[2]);
    printf("\n T axis position = %li", lpoint[3]);
}

void sendfile(void) /* Open file and send it to Unidex 14 */
{
    FILE *fp;
    char x;

```

```

printf("\nName of Unidex 14 command file to download (<CR> for no file):");
gets(fname);
if(strlen(fname) > 0)
{
    if((fp = fopen(fname, "r")) == NULL)
        printf("\nCan't open %s", fname);
    else
    {
        while(!feof(fp) == 0) /* while not end of file */
        {
            putcm(fgetc(fp)); /* get a char from file, send it to Unidex 14 */
            /* band aid - if DOS turns interrupt off, turn it back on */
            outp(IMREG, inp(IMREG) & ~ ILEVEL);
            while((x = getcm()) != NULL) /* while a char available from Unidex 14 */
            {
                putchar(x); /* print it */
            }
            checkstatus();
        }
        fclose(fp); /* close file when done */
        printf("%s transmitted.\n", fname);
    }
}
}

```

```

void interact(void) /* Sends keyboard entry to Unidex 14 and display its response */
{
    printf("\nInteractive mode, Enter Unidex 14 commands, Press Control-C to exit.\n");
    for (;;)
    {
        if (kbhit()) /* char available from keyboard? */
            putcm( getch() ); /* send it to Unidex 14 */
        if (mhit()) /* char available from Unidex 14 ? */
            putchar( getcm() ); /* get it and print it */
        checkstatus(); /* check U14's status register */
    }
}

```

```

void checkstatus(void) /* this routine decodes Unidex 14's status and displays it's info */
{
    extern char done_flag, error_flag;

    if (error_flag != 0)
    {
        if ((error_flag & CMDERR_BIT) != 0) /* command error */
            printf("\nUnidex 14's Status Register indicates a command error\n");
        else
        {
            if ((error_flag & ENC_BIT) != 0) /* encoder slip */
                printf("\nUnidex 14's Status Register indicates an Encoder Slip error\n");
            else
            {
                if ((error_flag & OVRT_BIT) != 0) /* limit switch is active */
                    printf("\nUnidex 14's Status Register indicates a limit switch is closed\n");
            }
        }
    }
}

```

```

    error_flag = 0; /* clear error_flag */
}

if(done_flag != 0) /* an ID has been encountered, print the done axis */
{
    printf("\nUnidex 14's Status Register indicates a DONE flag has been set\n");
    printf("AXES DONE: ");
    if(done_flag % 2 == 1)
        printf("x ");
    done_flag = done_flag / 2;
    if(done_flag % 2 == 1)
        printf("y ");
    done_flag = done_flag / 2;
    if(done_flag % 2 == 1)
        printf("z ");
    done_flag = done_flag / 2;
    if(done_flag % 2 == 1)
        printf("t ");
    printf("\nPress Control-Y, or type IC to clear Unidex 14's done status.\n");
    done_flag = 0; /* clear done flag */
}
}

char getcm(void)    /* Returns character from input buffer */
{
    char c;
    if(inxti == inxto)    /* no char available in buffer */
        return (0);
    else
    {
        c = inbuff[inxto + +]; /* get char. and incr. pointer */
        if(inxto == BSIZE) /* if end of buffer */
            inxto = 0; /* reset pointer */
        outp(CONTROL_REG, inp(CONTROL_REG) | IBF_BIT); /* enable U14 intr. */
        return (c);
    }
}

void putcm(char c)    /* Put character into output buffer */
{
    int i, j;

    i = onxti;
    j = i + 1;
    if(j == BSIZE)
        j = 0;
    while(j == onxto) /* wait till there's room for another chr. */
        ;
}

```

```

outbuff[i++] = c;
if( i == BSIZE )
    i = 0;
onxti = i;
outp( CONTROL_REG, inp( CONTROL_REG ) | TBE_BIT );
}

int mhit(void) /* returns non zero if input buffer has character */
{
    return(inxto - inxti);
}

interrupt far service() /* interrupt driven routine to communicate with Unidex 14 */
{
    char c;
    int i;

    c = inp(STATUS_REG) & (inp(CONTROL_REG) | 0x0f);
    /* get the status of interrupts that are enabled */

    if(c & DONE_BIT) /* if the DONE bit is true */
    {
        done_flag |= inp(DONE_REG); /* save done axis flags */
        error_flag |= (c & 0x0f); /* save error bits also */
        outp(DATA_REG, CLEAR); /* reset Unidex 14 status register DONE bit
                                without clearing flags shown in RA or
                                RI commands which ^Y does */
    }
    else
    if((c & TBE_BIT) == TBE_BIT) /* if Unidex 14 will accept a char */
    {
        if(onxto != onxti) /* do we have a char to send to Unidex 14 ? */
        {
            outp(DATA_REG, outbuff[onxto]); /* send char to Unidex 14 */
            if(++onxto == BSIZE) /* take care of wrap around */
                onxto = 0;
            if(onxto == onxti) /* if the buffer is now empty */
                /* turn off Unidex 14 interrupt so it won't keep asking for chars */
                outp(CONTROL_REG, inp(CONTROL_REG) & ~TBE_BIT);
        }
        else /* no char in buffer to send to Unidex 14 */
            /* turn off Unidex 14's interrupt so it won't keep asking for chars */
            outp(CONTROL_REG, inp(CONTROL_REG) & ~TBE_BIT);
    }
    if((c & IBF_BIT) == IBF_BIT) /* if Unidex 14 has a char to send to the PC */
    {
        inbuff[inxti] = inp(DATA_REG); /* get the char and put it in the buffer */
        if(++inxti == BSIZE) /* take care of wrap around */
            inxti = 0;
    }
}

```

```

    if((i = inxti + 1) == BSIZE) /* room for another char in the buffer? */
        i = 0;
    if(i == inxto)
        outp(CONTROL_REG, inp(CONTROL_REG) & ~IBF_BIT);
    /* no room in buffer, turn off interrupt so Unidex 14 cannot send more */
}

outp(ICNT, 0x20); /* send 8259 chip a return from interrupt */

/* PC/XT/AT interrupt control is edge sensitive only.
The following generates a rising edge if a second interrupt has occurred
prior to the first being reset. This can happen if more than one
source is enabled on the Unidex 14 as in this program. */

c = inp(CONTROL_REG);
outp(CONTROL_REG, c & ~IRQ_BIT); /* turn off interrupts */
outp(CONTROL_REG, c); /* and back on to create edge */
}

init(intvec) /* connect hardware interrupt IRQ5 to interrupt handler */
int (*intvec)();
{
    extern int intdosx();
    union REGS inregs, outregs;
    struct SREGS segregs;

    inregs.h.ah = 0x25; /* set interrupt vector DOS call */
    inregs.h.al = 13; /* interrupt vector level 5 */
    segregs.ds = FP_SEG(*intvec); /* send the vector */
    inregs.x.dx = FP_OFF(*intvec);
    intdosx(&inregs, &outregs, &segregs);
    outp(IMREG, inp(IMREG) & ~ILEVEL); /* enable interrupts */
    outp(CONTROL_REG, IBF_BIT | DONE_BIT | IRQ_BIT); /* turn on Unidex 14's interrupts */
}

void dma(int count, char *buffer)
{
    long addr;
    char far *message;

    message = buffer;
    addr = FP_SEG(message);
    addr = (addr << 4) + FP_OFF(message);
    printf("\n%x, %lx, %s\n", count, addr, buffer);
    initdma(count, addr);
}

```

```

void initdma(int count, long address)
{
    outp( MASKREG, 5 );    /* turn off channel 1 */
    outp( CONTROL_REG, inp( CONTROL_REG ) & ~ TBE_BIT ); /* TBE_BIT interrupts off */
    outp( DMAMODE, 0x49 ); /* single transfers, read memory */
    outp( BPF, 0 ); /* reset byte pointer F/F */
    outp( ADRREG, address & 0xff ); /* low order byte of address */
    outp( ADRREG, (address & 0xff00) >> 8 );
    outp( PGREG, (address & 0xf0000) >> 16 );
    count = 1; /* dma transfers one more than count */
    outp( COUNT, count & 0xff ); /* low order of byte count */
    outp( COUNT, (count & 0xff00) >> 8 );
    outp( CONTROL_REG, inp( CONTROL_REG ) & ~ DMARD ); /* prepare for writing */
    outp( CONTROL_REG, inp( CONTROL_REG ) | DMAEN ); /* enable the controller */
    outp( MASKREG, 1 ); /* turn on channel 1 */
    while( (inp( DMASTAT ) & 0x2) == 0 )
        ; /* wait for terminal count */
    outp( MASKREG, 5 ); /* turn off channel 1 */
    outp( CONTROL_REG, inp( CONTROL_REG ) & ~ DMAEN ); /* and turn off dma */
}

```

```

void home(void) /* this function will hardware home all axis */
{
    long result;
    writestring("BL13;WT9;BH13;"); /* send all axis home */
    printf("\nHoming");
    writestring("AX"); /* set x axis active */
    for(result=0; result < 100; result++) /* wait for home to start */
    {
        result = inbit(); /* read the inputs */
        while ((result & 32) != 0) /* wait for home completion */
            result = inbit(); /* read the inputs */
        writestring("ATLP0;"); /* set t axis pos. 0 */
        writestring("AZLP0;"); /* set z axis pos. 0 */
        writestring("AYLP0;"); /* set y axis pos. 0 */
        writestring("AXLP0;"); /* set x axis pos. 0 */
        printf("\nAt Home");
    }
}

```

```

long inbit(void) /* this function returns the state of the inputs */
{
    int x;
    long accum = 0; /* decimal equiv. accumulator */
    char result[20]; /* init. string */
    writestring("AXBX"); /* ask for input bit states */
    readlf(result); /* input the hex-ascii values */
}

```

```

for(x=0; x < 5; x++),
{ if (result[x] == '>'A') /* is it 'A' thru 'F' ? */
  result[x] = result[x]-7; /* pre-scale hex values */
  accum = accum + (power_16(5-x) * (result[x]-48)); /* calc. base ten value */
}
return(accum); /* return inputs in dec. */
}

```

```

long power_16(int power) /* this function returns 16 raised to the power of
    power. This is called by the inbit function only ! */

```

```

{
int i;
long p = 1;
for(i = 1; i < power; i++)
    p = p * 16;
return(p);
}

```

```

int fault(void) /* this function returns true '1' if any drive fault exists,
    on any servo axis, stepper drives do not have a fault
    output to Unidex 14. */

```

```

{
if((inbit() & 64) == 0) /* read the inputs */
    return(0); /* drive fault not present */
else
    return(1); /* drive fault present */
}

```

```

void inpos(char move[255]) /* this function starts a move, then waits for all
    servos, & steppers with 1MR bds. to come to rest.
    the move cmd string is expected to include a "go" cmd */

```

```

{
writestring(move); /* start the move */
printf("\nWaiting for drives to come to rest");
while(!(inbit() & 128)) /* wait for move to start */
    ;
while(inbit() & 128) /* wait for move to end */
    ;
}

```

```

void remote(char mode) /* this function controls the remote clk. & dir. inputs */
/* to the Unidex 14 power chassis. (L is PC control, */
{ /* R is remote clock & direction */
if((toupper(mode) == 'R')) /* convert mode chr. to upper case */
    writestring("AXBL12;"); /* set Remote mode */
else

```

```

    writestring ("AXBH12;"); /* set Local mode */
}

void getposition(long *position) /* this function returns a pointer to 4 long int's */
{
    char result[25];
    writestring("ATRP");      /* request position from Unidex 14 */
    readlf(result);
    position[3] = atol(result); /* find the t axis position */
    writestring("AZRP");      /* request position from Unidex 14 */
    readlf(result);
    position[2] = atol(result); /* find the z axis position */
    writestring("AYRP");      /* request position from Unidex 14 */
    readlf(result);
    position[1] = atol(result); /* find the y axis position */
    writestring("AXRP");      /* request position from Unidex 14 */
    readlf(result);
    position[0] = atol(result); /* find the x axis position */
}

void init_sys(void) /* this function initializes Unidex 14 system */
{
    int c;
    char result[40];
    writestring("AXEF");      /* kill echo if active */
    for (c=0; c < 3000; c++); /* delay till echo stops */
    {; }
    while ((inp(STATUS_REG) & IBF_BIT)) /* while there's a chr. in buffer */
        c = getcm(); /* input it */
    remote('L'); /* disable remote clk. & dir. */
    writestring("WY"); /* get Unidex 14 version number */
    readlf(result); /* input version number */
    printf ("\nVersion # is %s\n",result);
    /* writestring ("AAPA1,1,1,1;"); /* set high i mode during moves */
    writestring ("AXBH13;"); /* reset home cycle pulse */
}

void writestring(char cmd[255]) /* this function writes a string to Unidex 14 */
{
    int x = 0; /* set initial value */
    putcm('\x0d'); /* write a leading CR */
    while (cmd[x] != '\0') /* loop till all chr's sent */
    {
        putcm(cmd[x++]); /* write a chr. to Unidex 14 */
        checkstatus(); /* check Unidex 14's status register */
    }
}

```

```
char readchr(void) /* wait for a valid chr. and then return it */
{
char chr = 0;
while(chr == NULL) /* read till valid chr. */
chr = getcm(); /* get a chr. from Unidex 14 */
return(chr);
}

void readstring(int cnt, char *result) /* read cnt number of chr's and modify pointer to string */
{
int x;

for(x=0; x=(cnt-1); x++); /* read cnt number of chr's */
{ result[x] = readchr(); /* get a valid chr. */
}
result[x++] = '\0'; /* add NULL terminator */
}

void readlf(char *result) /* read chr's till a LF is received then begin inputting string until CR LF */
{
int temp = 0;

while(readchr() != CR) /* read till leading CR */
;
while((result[temp] = readchr()) != '\n') /* read till LF terminator */
{ temp++; /* build chr. string */
}
result[temp] = '\0'; /* add terminator */
while(readchr() != '\r') /* read till LF terminator */
;
}
```

SECTION 9-8: JOG.C

/* PROGRAM JOG.C

Version 1.01

This program allows the user to interact with the Unidex 14 motion control board. It uses the PC's interrupt line IRQ5 to communicate between the PC and Unidex 14. This program will have to be restarted if an RS (reset) command is sent to Unidex 14 because the interrupt control registers on the board are also reset causing the board to no longer create interrupts.

This program uses the computer's cursor keys to control the movement of two axes. The default are X and Y but the user may change this.

Portions of this program may be adapted by Unidex 14 users for application programs. No license is required.

This program was written using Microsoft C 5.1 Compiler.

RELEASED Aug. 27, 1991

*/

```
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include <bios.h>
#include <time.h>
#include <signal.h>
#include <graph.h>
```

/* Unidex 14 I/O port definitions */

```
#define DATA_REG    0x300
#define DONE_REG     0x301
#define CONTROL_REG  0x302
#define STATUS_REG   0x303
```

/* CONTROL and STATUS register bit definitions */

```
#define CMDERR_BIT 0x1
#define INIT_BIT   0x2
#define ENC_BIT    0x4
#define OVRT_BIT   0x8
#define DONE_BIT   0x10
#define IBF_BIT    0x20
#define TBE_BIT    0x40
```

```

#define IRQ_BIT    0x80

/* PC interrupt definitions */
#define ICNT      0x20    /* interrupt terminator byte for 8259 */
#define IMREG     0x21    /* interrupt mask register I/O address*/
#define ILEVEL    0x20    /* interrupt mask level 5 byte*/

/* character definitions */
#define CNTRL_C   3
#define CNTRL_Z   26
#define CLEAR     24
#define LF        10
#define CR        13

#define BSIZE     1000    /* input and output buffer size */

int  inxti, inxto, onxti, onxto; /* input and output buffer pointers */

char uaxis = 'y'; /*default axis controlled by up and down arrow keys*/
char laxis = 'x'; /*default axis controlled by left and right arrow keys*/
char left = '-'; /*default step direction controlled by left arrow key*/
char right = '+'; /*default step direction controlled by right arrow key*/
char up = '+'; /*default step direction controlled by up arrow key*/
char down = '-'; /*default step direction controlled by down arrow key*/
char joyexitmode = 'x'; /*axis mode when leaving joykey mode*/
long int stepvalue = 200; /*amount speed changes is multiplied by when a cursor key is pressed*/
long int speed = 200; /*current speed of joykey jog */
char *cmdstring = "joy keys not yet used"; /*joy key Unidex 14 command string*/

char  done_flag, error_flag;
char  inbuf[ BSIZE ], outbuf[ BSIZE ]; /* define buffers arrays */
char  fname[80]; /* file name buffer */
int   exit_handler(); /*traps control-C*/
char getc_u14();
int keyboard(); /*this routine decodes function and cursor keys and send commands to Unidex 14 */
interrupt far service();
unsigned char sameas(); /*returns complement cntrol or non-cntrol key for cursor keys*/

main()
{
    char inchar;
    inxti = inxto = onxti = onxto = 0; /* set buffer pointers to beginning*/
    done_flag = 0; /* clear done flag variable*/
    error_flag = 0; /* clear error flag variable*/

    _clearscreen(_GCLEARSCREEN);

```

```

printf("This program allows the user to control Unidex 14.\n");

init(service); /* initialize interrupts */
if(signal(SIGINT,exit_handler) == (int (*)()) -1) /*enable Cntrl-C trapping*/
{
    printf("Could not set up Control-C exit routine\n");
    abort();
}

/* sendfile(); /* get file name, open file, send it to Unidex 14 */

/*display anything returned from Unidex 14 resulting from the file sent to it*/
while(mhit() != 0) /* char available from Unidex 14 ? */
{
    inchar = getc_u14(); /* get char */
    putchar(inchar); /*display it */
}
interact(); /* interactive mode */
}

exit_handler()
{
    char ch;

    signal(SIGINT, SIG_IGN); /*turn off Control-C trapping while in this routine*/
    printf("\nPress y, Y or Esc to kill all axes' motors, any other key to continue.\n");
    ch = getch();
    if((ch == 'y') || (ch == 'Y') || (ch == 27))
    {
        putc_u14(' '); /*stop all axis*/
        putc_u14('k');
        putc_u14('l');
    }
    printf("Press y, Y or Esc to stop this program, any other key to continue.\n");
    ch = getch();
    putchar('\n');
    if((ch == 'y') || (ch == 'Y') || (ch == 27))
    {
        exit(0);
    }
    signal(SIGINT,exit_handler); /*turn on Control-C trapping*/
}

sendfile() /* Open file and send it to Unidex 14 */
{
    FILE *fp;

```

```
printf("Enter name of Unidex 14 command file to download. (
for no file)\n");
gets(fname);
if(strlen(fname) > 0)
{
    if ((fp = fopen(fname, "r")) == NULL)
        printf("Can't open %s\n", fname);
    else
    {
        while(!feof(fp) == 0)    /* while not end of file */
        {
           putc_u14(fgetc(fp)); /* get a char from file, send it to Unidex 14*/
            /* band aid - if DOS turns interrupt off, turn it back on */
            outp(IMREG, inp(IMREG) & ~ ILEVEL);
            while(mhbit())    /* while a char available from Unidex 14 */
            {
                putch(getc_u14()); /* print it*/
            }
            checkstatus();
        }
        fclose(fp);    /* close file when done */
        printf("The file \"%s\" has been sent to Unidex 14.\n", fname);
    }
}
```

```
interact() /* Sends keyboard entry to Unidex 14 and displays any response's */
```

```
{
    int x;
    int outchar = 0, inchar;

    printf("\nEnter Unidex 14 commands. Press F1 for help. Use cursor keys to control movement.\n");
    printf("If double characters are displayed enter \"ax ef\" to turn off Unidex 14's echo.\n");
    printf("Esc, ^Z or ^C to exit.\n\n");
    while (outchar != CNTRL_Z)
    {
        if (kbhit())    /* char available from keyboard? */
        {
            outchar = getch(); /* get char from keyboard */
            if(outchar == 27) /*if ESC is pressed ask if user wants to exit*/
                exit_handler();
            else
            if(outchar == 0) /*if a function or cursor key has been pressed*/
            {
```

```

    keyboard(); /*go to keyboard handler*/
}
else
{
    putchar(outchar); /*echo character to screen*/
    putc_u14(outchar); /* send it to Unidex 14 */
}
if(outchar == CR)
    putchar(LF); /* add linefeeds when return is pressed to
        avoid overstriking letters on the screen */
}

if (mhit()) /* char available from Unidex 14 ? */
{
    inchar = getc_u14(); /* get char */
    putchar(inchar); /*display it */
}

checkstatus(); /*see if any errors occurred or if done flag was set*/
}
}

help()
{
    _clearscreen(_GCLEARSCREEN);
    printf("SPECIAL PURPOSE KEYS ARE PROGRAMMED AS FOLLOWS:\n");
    printf("FUNCTION KEYS \ (keys not specified are not used)\n");
    printf("F1-This message is displayed.\n");
    printf("F2-sends U14 command \"rp\" to the board and displays values returned.\n");
    printf("F9-displays information about current programming of cursor keys.\n");
    printf("F10-allows user to change programmed value of cursor keys.\n");
    printf("\nCURSOR KEYS\n");
    printf("HOME (7)-Jog both axis defined by the left (4) and up (8) keys at the same\n
\n    time, at the same speed, resulting in a diagonal move.\n");
    printf("Up-arrow (8)-Jog its predefined axis in its predefined direction.\n");
    printf("    Power up defaults are Y axis and + direction.\n");
    printf("Pg Up (9)-Jog both axis defined by the right (6) and up (8) keys at the same\n
\n    time, at the same speed, resulting in a diagonal move.\n");
    printf("Left-arrow (4)-Jog its predefined axis in its predefined direction.\n");
    printf("    Power up defaults are X axis and - direction.\n");
    printf("Right-arrow (6)-Jog its predefined axis in its predefined direction.\n");
    printf("    Power up defaults are X axis and + direction.\n");
    printf("End (1)-Jog both axis defined by the left(4) and down (2) keys at the same\n
\n    time, at the same speed, resulting in a diagonal move.\n");
    printf("Down-arrow (2)-Jog its predefined axis in its predefined direction.\n");
    printf("    Power up defaults are Y axis and - direction.\n");

```

```

printf("Pg Dn (3)-Jog both axis defined by the right(4) and down(2) keys at the same\
\n    time, at the same speed, resulting in a diagonal move.\n");
printf("PRESS A KEY TO CONTINUE");
getch();
_clearscreen(_GCLEARSCREEN);
printf("The first time a cursor key is pressed the jog speed is the programmed speed.\n");
printf("Each time the same key is pressed or if the key is held down the speed will\n");
printf("increase by the programmed step value. Pressing any other key will stop\
\nmovement by sending an \"sa\" command to Unidex 14.\n\n");
printf("If the CONTROL key is held down while tapping the same cursor key\n");
printf("the speed is reduced by the step value. \n");
printf("\nOTHER SPECIAL KEYS\n");
printf("Ins (0)-Sends an \"sa\" command to Unidex 14, causing all axis to\
\n    decelerate to a stop.\n");
printf("Del (.)-Sends a \"kl\" command to Unidex 14, causing all axis to\
\n    stop instantly.\n");
printf("Esc - Asks user if they want to stop all motor movement and if they want\
\n    to terminate the program and performs these things if commanded to.\n");
printf("Control-C - Same function as Esc key.\n");
printf("Control-Z - Terminates program immediately, leaves Unidex 14 running.\n");
printf("\nEND OF HELP MESSAGE\n");
}

```

```

changejoyinfo()
{

```

```

    extern long int stepvalue; /*the size of the jumps in velocity when a cursor key is pressed*/
    extern long int speed; /*jog speed for joy keys*/
    extern char  udaxis; /*default axis controlled by up and down arrow keys*/
    extern char  laxis; /*default axis controlled by left and right arrow keys*/
    extern char  left; /*default step direction controlled by left arrow key*/
    extern char  right; /*default step direction controlled by right arrow key*/
    extern char  up; /*default step direction controlled by up arrow key*/
    extern char  down; /*default step direction controlled by down arrow key*/
    int ok,result;
    long int tempstep;
    char *string;
    char input,yn;

```

```

    printf("CHANGE CURSOR KEY PROGRAMMING\n");

```

```

    /*change step speed*/

```

```

    ok = 0;

```

```

    while(ok == 0)

```

```

    {
        printf("\nCurrently each time a cursor key is pressed the jog speed changes by: %ld\n",stepvalue);
    }

```

```

gets(string);
if(strlen(string) != 0)
{
    tempstep = atol(string);
    if( (tempstep > (long) 0) && (tempstep < 524000))
    {
        ok = 1;
        stepvalue = tempstep;
    }
    else
        printf("Step speed must be an integer greater than 0 and less than 524000.\n");
}
else
{
    printf("Step speed not changed\n");
    ok = 1;
}
}

/*change left-right axis*/
ok = 0;
while(ok == 0)
{
    printf("\nCurrent axis controlled by left (4) and right (6) cursor keys: %c\n",laxis);
    printf("Legal values are: x,y,z,t\n");
    printf("\n");
    printf("\n");
    printf("Axis to be controlled by left and right cursor keys (< CR > for no change):");
    input = getch();
    printf("\n");
    if(strchr("xyztXYZT",input) != 0)
    {
        ok = 1;
        laxis = input;
    }
    else
        if(input < 0x20) /*a <CR> or <LF> or other control char was entered so no change*/
            ok = 1;
        else
            printf("Illegal axis given\n");
}

/* change left direction */
ok = 0;
while(ok == 0)
{

```

```
printf("\nCurrent stepping direction controlled by left (4) cursor key: %c\n",left);
printf("Legal directions are: + and - \n");
printf("Enter direction to be controlled by left cursor key ( < CR > for no change):");
input = getche();
printf("\n");
if(strchr("+-",input) != 0)
{
    ok = 1;
    left = input;
}
else
if(input < 0x20) /*a < CR > or < LF > or or other control char was entered so no change*/
    ok = 1;
else
    printf("Illegal direction given\n");
}
/* change right direction */
ok = 0;
while(ok == 0)
{
    printf("\nCurrent stepping direction controlled by right (6) cursor key: %c\n",right);
    printf("Legal directions are: + and - \n");
    printf("Enter direction to be controlled by right cursor key ( < CR > for no change):");
    input = getche();
    printf("\n");
    if(strchr("+-",input) != 0)
    {
        ok = 1;
        right = input;
    }
    else
    if(input < 0x20) /*a < CR > or < LF > or or other control char was entered so no change*/
        ok = 1;
    else
        printf("Illegal direction given\n");
}

/*change up-down axis*/
ok = 0;
while(ok == 0)
{
    printf("\nCurrent axis controlled by up (8) and down (2) cursor keys: %c\n",udaxis);
    printf("Legal values are: x,y,z,t\n");
    printf("\n");
    printf("\n");
}
```

```

printf("Axis to be controlled by up and down cursor keys (< CR > for no change):");
input = getche();
printf("\n");
if(strchr("xyztXYZT",input) != 0)
{
    ok = 1;
    udaxis = input;
}
else
if(input < 0x20) /*a < CR > or < LF > or or other control char was entered so no change*/
    ok = 1;
else
    printf("Illegal axis given\n");
}

/* change up direction */
ok = 0;
while(ok == 0)
{
    printf("\nCurrent stepping direction controlled by up (8) cursor key: %c\n",up);
    printf("Legal directions are: + and - \n");
    printf("Enter direction to be controlled by up cursor key (< CR > for no change):");
    input = getche();
    printf("\n");
    if(strchr("+-",input) != 0)
    {
        ok = 1;
        up = input;
    }
    else
    if(input < 0x20) /*a < CR > or < LF > or or other control char was entered so no change*/
        ok = 1;
    else
        printf("Illegal direction given\n");
}

/* change down direction */
ok = 0;
while(ok == 0)
{
    printf("\nCurrent stepping direction controlled by down (2) cursor key: %c\n",up);
    printf("Legal directions are: + and - \n");
    printf("Enter direction to be controlled by down cursor key (< CR > for no change):");
    input = getche();
    printf("\n");
    if(strchr("+-",input) != 0)

```

```

{
    ok = 1;
    down = input;
}
else
if(input < 0x20) /*a < CR > or < LF > or other control char was entered so no change*/
    ok = 1;
else
    printf("Illegal direction given\a\n");
}
printf("\nReturned To Unidex 14 Control\n");

}

keyboard() /*this routine decodes function and cursor keys and sends
them to Unidex 14 */
{
    int length, letter, joymode, skip = 0;
    extern long int speed;
    unsigned int key, key1, key2, oldkey, opposkey;
    extern char *cmdstring;

    joymode = 0;

    key = getch(); /*get second part of function key code*/

    do /*while(joymode == 1)*/
    {
        switch (key)
        {
            case ',': /* F1 key, display help message*/
                help();
                if(joymode == 0) /*if not in the joy key mode*/
                {
                    skip = 1;
                }
                else
                    key = oldkey;
                break;

            case '<': /* F2 key, /*do an rp*/
                putc_u14(' ');
                putc_u14('r');
                putc_u14('p');
                while (mhit() != 0) /* char available from Unidex 14 */
                {

```

```

    putchar( getc_u14()); /* get char, display it */
}
if(joymode == 0) /*if not in the joy key mode*/
{
    skip = 1;
}
else
    key = oldkey;
break;

case 'C': /* F9 key, display current joykey values*/
    showjoyinfo();
    if(joymode == 0) /*if not in the joy key mode*/
    {
        skip = 1;
    }
    else
        key = oldkey;
    break;

case 'D': /* F10 key, change current joykey values*/
    changejoyinfo();
    if(joymode == 0) /*if not in the joy key mode*/
    {
        skip = 1;
    }
    else
        key = oldkey;
    break;

case 'G': /* home (7) key, move N.W. */
    joymode = 1;

    sprintf(cmdstring, " a%c jg%c%ld a%c jg%c%ld aa\n",laxis,left,speed,udaxis,up,speed);
    joyexitmode = 'a';
    break;

case 'H': /*up arrow (8) key, move N */
    joymode = 1;

    sprintf(cmdstring, " a%c jg%c%ld\n",udaxis,up,speed);
    joyexitmode = udaxis;
    break;

case 'I': /*pg up (9) key, move N.E. */
    joymode = 1;

```

```
    sprintf(cmdstring , " a%c jg%c%ld a%c jg%c%ld aa\n",laxis,right,speed,udaxis,up,speed);
    joyexitmode = 'a';
    break;

case 'K': /*left arrow (4) key, move W. */
    joymode = 1;

    sprintf(cmdstring , " a%c jg%c%ld\n",laxis,left,speed);
    joyexitmode = laxis;
    break;

case 'M': /*right arrow (6) key, move E. */
    joymode = 1;

    sprintf(cmdstring , " a%c jg%c%ld\n",laxis,right,speed);
    joyexitmode = laxis;
    break;

case 'O': /*end (1) key, move S.W. */
    joymode = 1;

    sprintf(cmdstring , " a%c jg%c%ld a%c jg%c%ld aa\n",laxis,left,speed,udaxis,down,speed);
    joyexitmode = 'a';
    break;

case 'P': /*down arrow (2) key, move S. */
    joymode = 1;

    sprintf(cmdstring , " a%c jg%c%ld\n",udaxis,down,speed);
    joyexitmode = udaxis;
    break;

case 'Q': /*pg dn (3) key, move S.E. */
    joymode = 1;

    sprintf(cmdstring , " a%c jg%c%ld a%c jg%c%ld aa\n",laxis,right,speed,udaxis,down,speed);
    joyexitmode = 'a';
    break;

case 119: /* CNTRL-home (7) key, move N.W. */
    joymode = 1;

    sprintf(cmdstring , " a%c jg%c%ld a%c jg%c%ld aa\n",laxis,left,speed,udaxis,up,speed);
    joyexitmode = 'a';
    break;
```

```
case 141: /* CNTRL-up arrow (8) key, move N */
    joymode = 1;

    sprintf(cmdstring, " a%c jg%c%ld\n",udaxis,up,speed);
    break;

case 132: /* CNTRL-pg up (9) key, move N.E. */
    joymode = 1;

    sprintf(cmdstring, " a%c jg%c%ld a%c jg%c%ld aa\n",laxis,up,speed,udaxis,right,speed);
    break;

case 115: /* CNTRL-left arrow (4) key, move W. */
    joymode = 1;

    sprintf(cmdstring, " a%c jg%c%ld\n",laxis,left,speed);
    break;

case 116: /* CNTRL-right arrow (6) key, move E. */
    joymode = 1;

    sprintf(cmdstring, " a%c jg%c%ld\n",laxis,right,speed);
    break;

case 117: /*CNTRL-end (1) key, move S.W. */
    joymode = 1;

    sprintf(cmdstring, " a%c jg%c%ld a%c jg%c%ld aa\n",laxis,left,speed,udaxis,down,speed);
    break;

case 145: /* CNTRL-down arrow (2) key, move S. */
    joymode = 1;

    sprintf(cmdstring, " a%c jg%c%ld\n",udaxis,down,speed);
    break;

case 118: /* CNTRL-pg dn (3) key, move S.E. */
    joymode = 1;

    sprintf(cmdstring, " a%c jg%c%ld a%c jg%c%ld aa\n",laxis,right,speed,udaxis,down,speed);
    break;

case 'R': /*ins (0) key, stop everything */
    joymode = 1;
    sprintf(cmdstring," sa");
```

```

    break;
case 'S': /*del (.) key, stop everything */
    joymode = 1;
    sprintf(cmdstring," kl");
    break;
default: /*all other non used function keys */
    joymode = 0;
    putch(key);
    printf(" = %d ",key);
    printf("KEY IS NOT USED\n");
    break;
}

if(joymode == 1) /*if a cursor control key was pressed*/
{
    /*send string to Unidex 14 */
    length = strlen(cmdstring);
    for(letter = 0 ; letter < length; ++ letter)
        putc_u14(cmdstring[letter]);

    while(!kbhit())
    {
        while (mhit() != 0)      /* char available from Unidex 14 ? */
        {
            putch( getc_u14()); /* get char, display it */
        }
        checkstatus();
    }
    key1 = getch(); /*get another keystroke*/
    if( key1 == 0) /*if it's a function key*/
    {
        key2 = getch(); /*get the rest of the keycode*/
        opposkey = sameas(key2);
        if((key2 == key) || (opposkey == key)) /*if the same key is pressed increase the speed*/
        {
            if( (opposkey >= 71) && (opposkey < 81)) /* if we are slowing down*/
            {
                speed -= stepvalue;
            }
            else /*we are speeding up*/
            {
                speed += stepvalue;
            }
            if(speed > 524000) /*cant go faster than this*/
                speed = 524000;
        }
    }
}

```

```

    if(speed < 1) /*cant go slower than this*/
        speed = 0;
        skip = 0;
        key = key2;
    }
    else
    if((key2 >= 59) && (key2 <= 68)) /*if its F1 to F10*/
    {
        /*don't change anything and save the last cursor key*/
        oldkey = key;
        key = key2;
        continue;
    }
    else /* a different cursor control key was pressed*/
    {
        putc_u14(' ');
        putc_u14('s');
        putc_u14('a');
        putc_u14(' '); /*stop all axis */
        putc_u14('a'); /*so it will be prepared to go in any direction*/
        putc_u14(joyexitmode); /*change default mode when leaving joykeys*/
        key = key2;
        speed = stepvalue;
        skip = 0;
    }

}

else /* a non cursor control, non-function key was pressed return it to main()*/
{
    if(skip == 0)
    {
        putc_u14(' ');
        putc_u14('s');
        putc_u14('a');
        putc_u14(' '); /*stop all axis */
        putc_u14('a'); /*so it will be prepared to go in any direction*/
        putc_u14(joyexitmode); /*go to default mode after leaving joykeys*/
        printf("a%c mode\n",joyexitmode);
    }
    joymode = 0;
    ungetch(key1);
    return;
}
}

while(joymode == 1);

```

```
return;
}

unsigned char sameas(key) /*look-up to find the code for the control or non-control version of cursor key*/
unsigned char key;
{
    unsigned char sameas;

    switch (key)
    {
        case 'G': /* home (7) key, move N.W. */
            sameas = 119;
            break;

        case 'H': /*up arrow (8) key, move N */
            sameas = 141;
            break;

        case 'I': /*pg up (9) key, move N.E. */
            sameas = 132;
            break;

        case 'K': /*left arrow (4) key, move W. */
            sameas = 115;
            break;

        case 'M': /*right arrow (6) key, move E. */
            sameas = 116;
            break;

        case 'O': /*end (1) key, move S.W. */
            sameas = 117;
            break;

        case 'P': /*down arrow (2) key, move S. */
            sameas = 145;
            break;

        case 'Q': /*pg dn (3) key, move S.E. */
            sameas = 118;
            break;

        case 119: /* CNTRL-home (7) key, move N.W. */
            sameas = 'G';
            break;
```

```
case 141: /* CNTRL-up arrow (8) key, move N */
    sameas = 'H';
    break;

case 132: /* CNTRL-pg up (9) key, move N.E. */
    sameas = 'I';
    break;

case 115: /* CNTRL-left arrow (4) key, move W. */
    sameas = 'K';
    break;

case 116: /* CNTRL-right arrow (6) key, move E. */
    sameas = 'M';
    break;

case 117: /*CNTRL-end (1) key, move S.W. */
    sameas = 'O';
    break;

case 145: /* CNTRL-down arrow (2) key, move S. */
    sameas = 'P';
    break;

case 118: /* CNTRL-pg dn (3) key, move S.E. */
    sameas = 'Q';
    break;

case 'R': /*ins (0) key, stop everything */
    sameas = 146;
    break;

case 'S': /*del (.) key, stop everything */
    sameas = 147;
    break;

default: /*all keys */
    sameas = 0;
/*    printf("sameas() passed illegal value:%c\n",key);*/
    break;
}
return(sameas);
}
```

showjoyinfo()

```

{
extern char *cmdstring; /*Unidex 14 commands created by keyboard();
extern unsigned long int speed; /*jog speed for joy keys*/
extern char udaxis; /*default axis controlled by up and down arrow keys*/
extern char laxis; /*default axis controlled by left and right arrow keys*/
extern char left; /*default step direction controlled by left arrow key*/
extern char right; /*default step direction controlled by right arrow key*/
extern char up; /*default step direction controlled by up arrow key*/
extern char down; /*default step direction controlled by down arrow key*/
extern char joyexitmode; /*axis mode when leaving joykey mode*/

printf("\n          JOYKEY INFORMATION\n\n");
printf("Last \"joy key\" command sent to Unidex 14: %s\n", cmdstring);
printf("Currently each time a cursor key is pressed the jog speed changes by: %ld\n\n",stepvalue);

printf("%c:axis controlled by up (8) and down (2) cursor keys.\n",udaxis);
printf("%c:stepping direction controlled by up (8) cursor key.\n",up);
printf("%c:stepping direction controlled by down (2) cursor key.\n\n",down);

printf("%c:axis controlled by left (4) and right (6) cursor keys.\n",laxis);
printf("%c:stepping direction controlled by left (4) cursor key.\n",left);
printf("%c:stepping direction controlled by right (6) cursor key.\n\n",right);

printf("a%c:axis control mode when leaving \"joykey\" mode.\n",joyexitmode);
printf("\n END OF JOYKEY INFORMATION\n");

}

checkstatus() /*this routine decodes Unidex 14's status and displays it */
{
extern char done_flag, error_flag;

if (error_flag != 0)
{
if ((error_flag & CMDERR_BIT) != 0) /*command error*/
printf("\nUnidex 14's Status Register indicates a COMMAND ERROR.\n");
else
if ((error_flag & ENC_BIT) != 0) /*encoder slip*/
printf("\nUnidex 14's Status Register indicates an ENCODER SLIP ERROR.\n");
else
if ((error_flag & OVRT_BIT) != 0) /*limit switch is active*/
printf("\nUnidex 14's Status Register indicates a limit switch has closed.\n");

error_flag = 0; /*clear error_flag*/
}
}

```

```

if(done_flag != 0) /*an ID has been encountered, print the done axis*/
{
    printf("\nUnidex 14's Status Register indicates a DONE flag has been set.\n");
    printf("AXES DONE: ");
    if(done_flag % 2 == 1)
        printf("x ");
    done_flag = done_flag / 2;
    if(done_flag % 2 == 1)
        printf("y ");
    done_flag = done_flag / 2;
    if(done_flag % 2 == 1)
        printf("z ");
    done_flag = done_flag / 2;
    if(done_flag % 2 == 1)
        printf("t ");
    printf("\nPress Cntrl-Y or type IC to clear Unidex 14's done status.\n");
    done_flag = 0; /*clear done flag */
}
}

char getc_u14() /* Returns character from Unidex 14's buffer */
{
    char c;

    if(inxti == inxto) /* no char available in buffer */
        return (0);
    else
    {
        c = inbuff[inxto + +]; /* get char that pointer points to, incr. pointer*/
        if(inxto == BSIZE) /* if we are pointing to the end of the buffer */
            inxto = 0; /* wrap pointer around to beginning of buffer */
        outp(CONTROL_REG, inp(CONTROL_REG) | IBF_BIT); /* enable Unidex 14
            to interrupt PC to put more chars. in buffer */
        return (c);
    }
}

putc_u14(c) /* Puts character into output buffer to Unidex 14 */
char c;
{
    int i, j;

    i = onxti; /* set local pointer to global input pointer of output buffer*/
    j = i + 1;
    if(j == BSIZE) /*if next pointer points to the end, wrap pointer to start*/

```

```

    j = 0;
    while(j == onxto) /* while there is no room in the buffer for more */
    ; /*wait until Unidex 14's interrupt routine makes room in buffer*/
    outbuf[i++] = c; /* get character i points to and increment i */
    if(i == BSIZE) /* if i points to the end of buffer wrap it around */
        i = 0;
    onxti = i; /*set global pointer to local pointer's value*/
    outp(CONTROL_REG, inp(CONTROL_REG) | TBE_BIT); /* enable Unidex 14 to interrupt
        PC so it can get chars from the buffer*/
}

mhit() /* returns non zero if input buffer has character */
{
    return(inxto - inxti);
}

interrupt far service() /* interrupt driven routine to communicate with Unidex 14 */
{
    char c;
    int i;

    /* get the status of interrupts that are enabled */
    c = inp(STATUS_REG) & (inp(CONTROL_REG) | 0x0f);

    if(c & DONE_BIT) /* if the DONE bit is true */
    {
        done_flag |= inp(DONE_REG); /* save done axis flags */
        error_flag |= (c & 0x0f); /* save error bits also */
        outp(DATA_REG, CLEAR); /* reset Unidex 14 done status */
    }
    else
    if((c & TBE_BIT) == TBE_BIT) /* if Unidex 14 board will accept a char */
    {
        if(onxto != onxti) /* do we have a char to send to Unidex 14 ? */
        {
            outp(DATA_REG, outbuf[onxto]); /* send char to Unidex 14 */
            if(++onxto == BSIZE) /* take care of wrap around */
                onxto = 0;
            if(onxto == onxti) /* if the buffer is now empty */
                /* turn off Unidex 14's interrupt so it won't keep asking for chars */
                outp(CONTROL_REG, inp(CONTROL_REG) & ~TBE_BIT);
        }
        else /* no char in buffer to send to Unidex 14 */
            /* turn off Unidex 14's interrupt so it won't keep asking for chars */
            outp(CONTROL_REG, inp(CONTROL_REG) & ~TBE_BIT);
    }
}

```

```

}
if((c & IBF_BIT) == IBF_BIT) /* if Unidex 14 has a char to send to the PC*/
{
    inbuf[inxti] = inp(DATA_REG); /* get the char and put it in the buffer*/
    if(++inxti == BSIZE) /* take care of wrap around */
        inxti = 0;
    if((i = inxti + 1) == BSIZE) /* room for another char in the buffer? */
        i = 0;
    if(i == inxto)
        outp(CONTROL_REG, inp(CONTROL_REG) & ~IBF_BIT);
    /*no room in buffer, turn off interrupt so Unidex 14 cannot send more*/
}
outp(ICNT, 0x20); /* send 8259 chip a return from interrupt */

/* PC/XT/AT interrupt control is edge sensitive only.
The following generates a rising edge if a second interrupt has occurred
prior to the first being reset. This can happen if more than one
source is enabled on Unidex 14 as in this program. */

c = inp(CONTROL_REG);
outp(CONTROL_REG, c & ~IRQ_BIT); /* turn off interrupts */
outp(CONTROL_REG, c); /* and back on to create edge */
}

init(intvec) /*connect hardware interrupt IRQ5 to interrupt handler */
int (*intvec)();
{
    extern int intdosx();
    union REGS inregs, outregs;
    struct SREGS segregs;

    inregs.h.ah = 0x25; /* set interrupt vector DOS call */
    inregs.h.al = 13; /* interrupt vector level 5 */
    segregs.ds = FP_SEG(*intvec); /* send the vector */
    inregs.x.dx = FP_OFF(*intvec);
    intdosx(&inregs, &outregs, &segregs);
    outp(IMREG, inp(IMREG) & ~ILEVEL); /* unmask Unidex 14's interrupts */
    outp(CONTROL_REG, IBF_BIT | DONE_BIT | IRQ_BIT); /*turn on Unidex 14 interrupts*/
}

```

SERVICE AND REPAIR

If necessary, any on-site service should be performed by an experienced electronic technician, preferably one trained by Aerotech, Inc.

SHIPMENT

The procedure for shipping equipment back to Aerotech described below, pertains to warranty as well as non-warranty returns.

1. Before shipping any equipment back to Aerotech, the person making the return must call ahead for a "*Return Authorization Number*".
- 2 .The equipment being returned must be encased in a proper cushioning material and enclosed in a cardboard box.

Call for a "Return Authorization Number" if it is necessary to ship any part to the factory.

NOTE: Aerotech is not responsible for equipment damage that is due to improper packaging.

Aerotech Service offices are listed on the following page. For service and information, contact the office servicing your area.

AEROTECH SERVICE CENTERS

World Headquarters

AEROTECH, INC.

101 Zeta Drive

Pittsburgh, PA 15238

Phone (412) 963-7470

FAX (412) 963-7459

TLX (710) 795-3125

AEROTECH LTD.

3 Jupiter House, Calleva Park

Aldermaston

Berkshire RG7 4QW England

Phone (07356) 77274

TLX 847228

FAX (07356) 5022

AEROTECH GMBH

Neumeyerstrasse 90

8500 Nuernberg 10

West Germany

Phone (0911) 521031

TLX 622474

FAX (0911) 521235



Warranty and Field Service Policy

Aerotech, Inc. warrants its products to be free from defects caused by faulty materials or poor workmanship for a minimum period of one year from date of shipment from Aerotech. Aerotech's liability is limited to replacing, repairing or issuing credit, at its option, for any products which are returned by the original purchaser during the warranty period. Aerotech makes no warranty that its products are fit for the use or purpose to which they may be put by the buyer, whether or not such use or purpose has been disclosed to Aerotech in specifications or drawings previously or subsequently provided, or whether or not Aerotech's products are specifically designed and/or manufactured for buyer's use or purpose. Aerotech's liability on any claim for loss or damage arising out of the sale, resale or use of any of its products shall in no event exceed the selling price of the unit.

Laser Product Warranty

Aerotech, Inc. warrants its laser products to the original purchaser for a minimum period of one year from date of shipment. This warranty covers defects in workmanship and material and is voided for all laser power supplies, plasma tubes and laser systems subject to electrical or physical abuse, tampering (such as opening the housing or removal of the serial tag) or improper operation as determined by Aerotech. This warranty is also voided for failure to comply with Aerotech's return procedures.

Return Products Procedure

Claims for shipment damage (evident or concealed) must be filed with the carrier by the buyer. Aerotech must be notified within (30) days of shipment of incorrect materials. No product may be returned, whether in warranty or out of warranty, without first obtaining approval from Aerotech. No credit will be given nor repairs made for products returned without such approval. Any returned product(s) must be accompanied by a return authorization number. The return authorization number may be obtained by calling an Aerotech service center. Products must be returned, prepaid, to an Aerotech service center (no C.O.D. or Collect Freight accepted). The status of any product returned later than (30) days after the issuance of a return authorization number will be subject to review.

Returned Product Warranty Determination

After Aerotech's examination, warranty or out-of-warranty status will be determined. If upon Aerotech's examination a warranted defect exists, then the product(s) will be repaired at no charge and shipped, prepaid, back to the buyer. If the buyer desires an air freight return, the product(s) will be shipped collect. Warranty repairs do not extend the original warranty period.

Returned Product Non-Warranty Determination

After Aerotech's examination, the buyer shall be notified of the repair cost. At such time the buyer must issue a valid purchase order to cover the cost of the repair and freight, or authorize the product(s) to be shipped back as is, at the buyer's expense. Failure to obtain a purchase order number or approval within (30) days of notification will result in the product(s) being returned as is, at the buyer's expense. Repair work is warranted for (90) days from date of shipment. Replacement components are warranted for one year from date of shipment.

Rush Service

At times, the buyer may desire to expedite a repair. Regardless of warranty or out-of-warranty status, the buyer must issue a valid purchase order to cover the added rush service cost. Rush service is subject to Aerotech's approval.

On-Site Warranty Repair

If an Aerotech product cannot be made functional by telephone assistance or by sending and having the customer install replacement parts, and cannot be returned to the Aerotech service center for repair, and if Aerotech determines the problem could be warranty-related, then the following policy applies.

Aerotech will provide an on-site field service representative in a reasonable amount of time, provided that the customer issues a valid purchase order to Aerotech covering all transportation and subsistence costs. For warranty field repairs, the customer will not be charged for the cost of labor and material. If service is rendered at times other than normal work periods, then special service rates apply.

If during the on-site repair it is determined the problem is not warranty related, then the terms and conditions stated in the following "On-Site Non-Warranty Repair" section apply.

On-Site Non-Warranty Repair

If an Aerotech product cannot be made functional by telephone assistance or purchased replacement parts, and cannot be returned to the Aerotech service center for repair, then the following field service policy applies.

Aerotech will provide an on-site field service representative in a reasonable amount of time, provided that the customer issues a valid purchase order to Aerotech covering all transportation and subsistence costs and the prevailing labor cost, including travel time, necessary to complete the repair.

AEROTECH, Inc., 101 Zeta Drive, Pittsburgh, Pennsylvania 15238

Phone (412) 963-7470 • TWX 710-795-3125 • FAX (412) 963-7459

INDEX

A

Auxiliary Control,4-18

C

Command Structure

- Axis Specification,8-3
- Command Summary,8-60
- Constant Velocity Contouring,8-55
- Encoder Status,8-50
- Encoder Tracking,8-49
- Home,8-31
- Initialization,8-31
- Loop Control,8-27
- Move Execution,8-23
- Move Specifications,8-16
- Move Synchronization,8-34
- Move Termination,8-25
- Position Maintenance,8-44
- Slip/Stall Detection,8-48
- System Control,8-6
- System Status Request,8-39
- User I/O,8-12
- User Units,8-43
- Velocity Staircase,8-52

Connections,4-5

- Auxiliary Control,4-16
- Encoder,4-7
- I/O,4-19
- Limit Switch,4-7
- Marker,4-7,4-13
- Power,4-29
- Serial Output,4-22
- Cosine Ramp,2-20

D

Demonstration Software,9-1

- Basic,9-3
- C,9-35
- Jog.C,9-41
- Pascal,9-11
- Poll.C,9-27
- Readme,9-2

Description,1-1

Dimensions,7-12

DMA,2-6,3-2,3-5

Drive Chassis,5-1

Drive Modules,6-1

Aerodrive

- Adjustments,6-54
- Analog Signal,6-59
- Circuit Description,6-48
- Encoder Parameters,6-53
- Home Direction,6-51
- Home Limit,6-51
- Home Reference,6-48
- Limit Switch,6-51
- Marker Input,6-52
- Operation,6-48
- Specifications,6-62
- Torque/Speed,4-63

DC Servo Drives

- Adjustments,6-34
- Analog Signal,6-41
- Circuit Description,6-26
- Encoder Parameters,6-33
- Fuse Ratings,6-29
- Home Direction,6-31
- Home Limit,6-31
- Home Reference,6-30
- Limit Switch,6-33
- Marker Input,6-32
- Operation,6-26
- Specifications,6-45
- Tachometer,6-29
- Torque/Speed,4-46

Stepping Drives,6-1

- Circuit Description,6-1
- Direction,6-5,6-18
- Home Clock,6-8,6-21
- Home Limit,6-3,6-15
- Home Reference,6-2,6-14
- Limit Switch,6-5,6-17
- Lo/Hi Current,6-8,6-21
- Personality Module,6-7,6-20
- Marker Buffer,6-4,6-16
- Resolution,6-6,6-18
- Specifications,6-12
- Torque/Speed,6-13

H

Home Marker,4-13

I

I

Interrupts,2-6,3-2,3-5

Interface Board

Jumpers,2-12

Internal Structure,5-1

Introduction,2-1

I/O

Address,2-5,3-3

Channel,3-1,3-4

Clock/Osc,3-3

Data Bus,3-1

Input,2-7

Memory,3-1

Power Supply,3-8

Output,2-7

Registers,3-6

Reset,3-3

Pull-Jumpers,2-6

L

Linear Ramps,2-18

Limit Inputs,2-3

Limit Switch,7-11

M

Motors,7-1

P

Packaging,1-1

Parabolic Ramp,2-19

Power Supply,6-64

Program Examples,8-64

R

Rear Panel,4-2

S

Software

Installation,2-9

Specifications, 1-3,1-4,1-5

T

Theory of Operation,2-2

U

U14C

Description,5-7

Jumpers,2-3

Installation,2-8

Motor Control Connector,2-10

Schematics,5-8

V

Velocity Profiles,2-17