Addendum to 400030-139 (Latest Revision) For Use With The SS2000-PCi-2 Programmable Step Motor Controller

The following pages have been revised for use with the SS2000PCi-2.

1. Replace Section 1 (Pages 1 – 4) with the new Section 1.

   Revised/added references to SS2000PCi-2.

2. Replace Section 3 (Pages 9 – 30) with the new Section 3.

   Revised Mechanical Outline diagram
   Revised Hardware Connection Descriptions
   Revised XWC description and diagram

3. Replace Sections 4.1, 4.2 and 4.2.1 (Page 32) with the new page provided.

   Revised Mechanical and Electrical Specifications

4. Replace Sections 5.1 – 5.1.4.1) (Page 38) with the new page provided.

   Added references to SS2000PCi-2

5. Add new pages 156-159 to Section 7

   Added two new sections,
   7.4 Two Independent Axis Control Programming Example (Using AUX Port)
   7.5 Two Independent Axis Control Programming Example (Using I/O Port)

6. Add new Section 10 Host – Slave Control and Peer-to-Peer Control of Axis 1 & 2.

   This section describes in detail how to wire and communicate between the 2 independent axis' of the SS2000PCi-2 in two configurations, Host – Slave and Peer-to-Peer.

Superior Electric
383 Middle Street • Bristol, CT 06010
(860)585-4500 • Fax:(860)589-2136
web site:www.superiorelectric.com

Printed in U.S.A.

Section 1

Introduction

1.1 - HOW TO USE THIS MANUAL

Congratulations on the purchase of your new Superior Electric SLO-SYN® WARPDRIVE™ family motion control product! The SS2000PCi-2 programmable motion controller is a full-featured and flexible product, yet it is fairly simple to apply it to your machine control application. This manual is designed to guide and assist you through the installation, programming, and operation of the controller/drive. If you're reading this, you understand the importance of familiarizing yourself with how this product should be installed and operated. We strongly recommend that you read through this manual until you are comfortable with electrical connections and operating concepts of the unit. Also, for your safety, we strongly recommend that you read "Section 2 - Important Safety Information" first, then read the "Quick Start Installation Guide" section. This will provide you with the basics on how to properly wire and connect the unit into your system. From there you can move on to the "Motion Controller Programming Interface" and "Software Reference Guide" sections to learn how to program your controller/drive to suit your application. "The "Glossary" section describes the terms most commonly used in this manual. Detailed technical information is provided in the "Hardware Specifications" section.

1.2 – FEATURES AND FUNCTIONS

The SS2000PCi-2 two-axis step motor controller features two fully programmable digital indexers (controllers) capable of independently operating two step motors, pulse-and-direction drives non-simultaneously. This single package occupies less panel space for applications that require 2 independent or 2 coordinated (non-simultaneous) axes of motion. Each indexer is a powerful controller which allows motion programming using the Motion Controller Programming Interface (MCPI). The MCPI is a Windows® based Graphical User Interface (GUI) which runs on a PC and facilitates system programming in an easy to use BASIC like language.

1) <u>Microcontroller Based Digital Controller Card</u>.

Each of the two controller/indexer circuit cards is based on a sophisticated digital microcontroller chip. The microcontroller performs all necessary tasks for executing complex user programs including control of digital inputs and outputs (I/O), stepper motor current profiles, two serial communications ports, drive section interface, digital encoder inputs for closed loop stepper, etc.

Some of the key features of the controller are:

- High-performance motion controller uses ~~16-bit,~~ 16-MHz microprocessor
- Surface-mount construction utilizing custom integrated circuits
- Easy setup and programming with Windows interface and BASIC-like language
- Simpler program construction – user specifies own motion units, e.g. inches, the programming environment automatically converts to motor "steps"
- Feature-rich command set, with over 85 functions in the following groups: Motion, Trajectory Parameters, Drive Parameters, I/O Control, Timer, Program Flow Control, Interrupts, Boolean/Relational Operators, String Handling, Variables, and Arithmetic
- Two independent serial ports: Host port RS232 or RS485; Auxiliary port is RS485; selectable communication rates up to 38.4 kbaud
- Programmable Inputs: 8 optically isolated 5-24 VDC; 8 more logic inputs (or used for BCD switches)
- Programmable Outputs: 4 optically isolated 5-24 VDC, 250 mA; 4 more "sinking" open collector (or used for BCD switches)
- Built-in 12 VDC power supply for opto-coupled I/O
- Analog input: 0-10 Volts, 10-bit resolution with multiple programmable functions
- I/O on shielded 25-pin "D" connector; optional terminal-strip adapter available
- Encoder input of up to 2 million counts per second on 9-pin "D" connector
- Closed-loop modes for stall detection, position verification, and correction
- 3 LED indicators for Power, Fault, and Motion Busy

2) <u>SS2000PCi-2 Power Supply</u>

- Wide range AC input 95 – 265 VAC, 50/60 Hz
- Fused AC Input
- Built-in Line Filter
- Current Foldback on DC Power Ouputs

1.3 - WHAT YOU NEED TO KNOW FIRST

This instruction manual is written in a simple and easy-to-follow format that should be suitable for both new and experienced motion control users. In order to get the most out of your SS2000PCi-2 Programmable Motion Controller, we assume the user will be knowledgeable in the following areas:

1. Basic electrical and electronic skills, including preparing and following an equipment wiring diagram or schematic.
2. The basics of motion control system applications, such as torque, speed, move distances, how to structure a motion task into move segments and input/output control.
3. Some familiarity with elementary computer programming, including defining the problem to be solved and coding it in a computer language.

1.4 - CONVENTIONS USED IN THIS MANUAL

1. Motor rotation direction (CW and CCW) is properly oriented when viewing the motor from the end opposite the mounting flange end of motor.
2. Please refer to Section 9 “Glossary” for detailed descriptions of terms such as "sink and source I/O", various motion terms, etc.

1.5 - HOW TO CONTACT US

Although this manual represents a detailed compilation of information regarding your SLO-SYN WARPDRIVE control product, sometimes questions may arise which will require that you contact us. You now have a few options available to you when you need information regarding your product or its application:

1. **On the Internet at www.superiorelectric.com.** Our multimedia enabled web site offers you information such as:

   - Free Software
   - Software Updates
   - TechFax fax on demand documents (1-800-234-3369)
   - HTML Product Selector
   - HTML Brand Selector
   - Product News
   - Links
   - Sales and Distribution Information
   - Product information and specifications
   - Literature Requests
   - Technical Support E-mail
   - Many more features

2. **By Phone.** You may reach us by phoning our Motion Control Applications Engineering Department at telephone (800) 787-3532. We may be reached between the hours of 8:00 am and 5:00 PM (Eastern Time), Monday through Friday. Technical personnel are available to assist you in getting your application up and running as efficiently as possible.

(This page intentionally left blank)

Section 3

Quick Start Installation Guide

3.1 - Step-by-Step Start-Up Procedure

The SS2000PCi-2 step motor controller is a sophisticated and versatile product. Setting up the system, however, can be simple and straight-forward if the proper steps are followed. Please use the step-by-step set up guide below.

1) Bench Set Up.

Before connecting your SS2000PCi-2 and drive to your motor and mechanical system or machine, we recommend that you "bench test" the system. This will allow you to become familiar with the wiring, programming and operation of the system before installing it into your machine. This may also prevent inadvertent damage to your mechanical system if you make programming errors which cause unexpected motion. The bench set up can be used to perform simple motions with an unloaded motor. To perform a bench test, do the following:

a) **Wire it up**. Read Section 3.5 Wiring Diagrams, and connect the AC power, I/O and other required signals per the wiring diagrams and instructions. **BE SAFE!!** Do not apply AC power to the unit until you are sure of all connections. Initially, there is no need to connect all of the wiring of your system together. Wire the AC line input, motor drive and HOST communication ports. This will be all you need to establish communications to the unit and perform simple motion.

b) **Load Software.** You will need to use a PC to program the unit according to your requirements. First you must load the MCPI software onto the PC from the floppy disks provided with your unit. Simply insert disk #1 and run the file SETUP.EXE. Once the software is loaded, run it by double clicking on the MCPI icon. See Section 5 for more details on the MCPI installation process.

c) **Create your Project**. You can now create your new **Project.** Your Project will contain **Configuration** information for your particular system, and also your program **Task** which holds the user program written in BASIC-like language. Read section 5 of this manual, and then step through the Configuration folders and enter the appropriate data for your system, saving the configuration when you are done. Note that for this exercise, the original default settings should work fine. Don't forget to set up the serial port for your PC to the correct port number and baud rate.

<u>HINT:</u> Motion is commanded in **User Units.** The System folder in the Configuration allows you to enter User Units per motor revolution. Initially, it is easiest to set this to 1. This will mean that move distances are in motor revolutions (e.g. movei=1 moves one revolution), speeds will be in revs/sec, and accelerations will be in revs/sec/sec. Later this can be changed (e.g. to allow programming in inches on a lead screw) to allow ease of programming once the motor is installed into the mechanical system. See the System Folder section of this manual for other examples. **All** move distances, speeds, and accelerations (or decelerations), and encoder information are provided in User Units, so be sure you understand this before continuing.

d) **Compile and Download** the project into the unit using the command buttons of the MCPI. Note that initially, you can leave the Task blank and command motion using the **Host Commands.** Host commands are entered in **Terminal Mode** from the MCPI. Enter the terminal mode using the appropriate command button on you screen.

e) **Make it move!** Now that you have compiled and downloaded your project into the unit, you are ready to make the motor move. First you must enter the speed at which you wish the motor to turn, such as 1 rev/sec. Do this by typing speed=1 <CR> (the <CR> means the Return or Enter key). Now enter the acceleration, for example 50 revs/sec/sec by typing accel=50<CR>. Set the deceleration to match by typing decel=50<CR>. After each entry, the controller should respond with a ">" prompt indicating that it has accepted your command. With the motor secured to the bench, you can now command a move. To command an incremental move of 10 revolutions type movei=10<CR>. The motor should now move 10 revolutions. If it does not, check your wiring and verify your configuration settings. In addition, check the motor direction to insure it meets your requirements. The motor direction can be reversed in the System folder if necessary.

f) **Write a BASIC Program.** Now that you have made a simple move, you are ready to write your Task in the MCPI BASIC-like language. Refer to section 6 for a complete description of all of the **Program Commands.** You can start by opening your Task and entering the commands. First, let's enter the exact same commands that you used in the Terminal HOST mode. Enter the speed, accel, decel, and movei commands as you did in step e) above. You must enter two more commands to tell the unit that the program is done after it performs the move. Type waitdone<CR> and End<CR> as the last lines of the program. Since your program has changed, you must compile and download it into the unit again for the changes to take effect. If you receive compilation errors, check your spelling and syntax with the information in section 6.

g) **Execute the Program.** . From the Terminal screen, click on the RUN button to make the motor move 10 revolutions. If desired you can now add lines to the program to perform more sophisticated motion. For example, try typing REAL x <CR> as the first line of your program. This will *declare* x as a **REAL variable**. See sections 5 and 6 for discussions regarding variables. On the next line, type x=10 <CR>. This assigns the REAL variable x a value of 10. Change the movei=10 line to movei=x. Now the motor will move whatever distance has been assigned to x. Recompile and download your program, then run it. It should operate the same as before, but now the program is now using x as the move distance in place of 10 as before. Change the value of x to different distance values to verify that it works correctly.

h) **Expand the Program and Debug it.** Now that you have written a simple program, you can add more complexity by adding more commands. You can do complex looping, access I/O, and motion functions as required. It will be helpful now to use the **DEBUG** feature of the MCPI. Again, refer to section 5 for a detailed description of the debug mode. If you compile your program in Debug Mode, you can enter the debug screen as your program runs and step through your code to verify proper operation. Once the code is functioning correctly, you should re-compile in Release Mode as this will speed up program execution.

2) Installation into Mechanical System

Once you have tested everything out in a controlled environment, you may complete the installation into your system. This will require making all the necessary wiring connections for limit switches, additional I/O, analog inputs, encoder, etc. **Start simple!!** Just as you started with a simple move on the bench, you should start simple here as well, slowly adding complexity as you debug your code and gain more confidence in programming. You may use the Debug Mode to help in this process. Once you have the program running the way you want, you can disconnect the HOST computer and use the RUN switch input or Program Autostart feature in the Configuration to run your program without a computer attached.

3.1.1 - Switch Settings

Caution

Before mounting and wiring your Slo-Syn Controller, the switches that govern various operating features should be checked or set to their proper positions for your application.

Warning

NEVER change the switch settings with the unit powered ON. Risk of physical injury or damage to the unit may result.

3.1.2 - Baud Rate and Unit ID Switch

The Baud Rate switch is accessible through the top of the unit and has two positions, 9600 or User Baud. According to the switch position, upon unit power up or RESET, the baud rate is set to either 9600 or the User Baud rate. If the switch is in the User position, the unit baud rate is set to the baud rate parameter defined in the downloaded project. If the switch is in the 9600 position, the baud rate will be forced to 9600 **regardless of the project configuration.**

It is possible to communicate to multiple units over the same RS-485 transmission lines. To accomplish this, the SS2000PCi-2 supports daisy chain wiring of from 2 to 32 units. **All units MUST have their HOST communications port set to RS-485 mode for daisy chaining to function properly. Insure that the power is off when changing the switch position.** To change the Host port communications mode, slide the RS-232/RS-485 selector switch to the appropriate location. The switch is accessible through the top of the unit near the BCD I/O port. All units must also be set to the same baud rate.

Further wiring details are included in Section 3.5 Wiring Diagrams. Note that **RS-232 daisy chaining is NOT supported,** and RS-232 signals should NOT be connected to the Host port when it is in RS-485 mode.

The Host command <nn allows different modes for daisy chain communications. Refer to the Host Command section for a detailed description of the daisy chain commands including their syntax and usage.

Each unit on the daisy chain must have a unique identification number (ID) to eliminate transmitter conflicts on the RS485 port. Five dip switches are provided for selecting the unit ID (1 – 32). They are accessible through the top rear of the unit. One and only one unit MUST have ID 1. The switch positions are only decoded at power-up. Do not change the switches with the power on.

The unit ID 's are decoded as follows:

ID Num.	SW-1	SW-2	SW-3	SW-4	SW-5
1	ON	ON	ON	ON	ON
2	ON	ON	ON	ON	OFF
3	ON	ON	ON	OFF	ON
4	ON	ON	ON	OFF	OFF
5	ON	ON	OFF	ON	ON
6	ON	ON	OFF	ON	OFF
7	ON	ON	OFF	OFF	ON
8	ON	ON	OFF	OFF	OFF
9	ON	OFF	ON	ON	ON
10	ON	OFF	ON	ON	OFF
11	ON	OFF	ON	OFF	ON
12	ON	OFF	ON	OFF	OFF
13	ON	OFF	OFF	ON	ON
14	ON	OFF	OFF	ON	OFF
15	ON	OFF	OFF	OFF	ON
16	ON	OFF	OFF	OFF	OFF
17	OFF	ON	ON	ON	ON
18	OFF	ON	ON	ON	OFF
19	OFF	ON	ON	OFF	ON
20	OFF	ON	ON	OFF	OFF
21	OFF	ON	OFF	ON	ON
22	OFF	ON	OFF	ON	OFF
23	OFF	ON	OFF	OFF	ON
24	OFF	ON	OFF	OFF	OFF
25	OFF	OFF	ON	ON	ON
26	OFF	OFF	ON	ON	OFF
27	OFF	OFF	ON	OFF	ON
28	OFF	OFF	ON	OFF	OFF
29	OFF	OFF	OFF	ON	ON
30	OFF	OFF	OFF	ON	OFF
31	OFF	OFF	OFF	OFF	ON
32	OFF	OFF	OFF	OFF	OFF

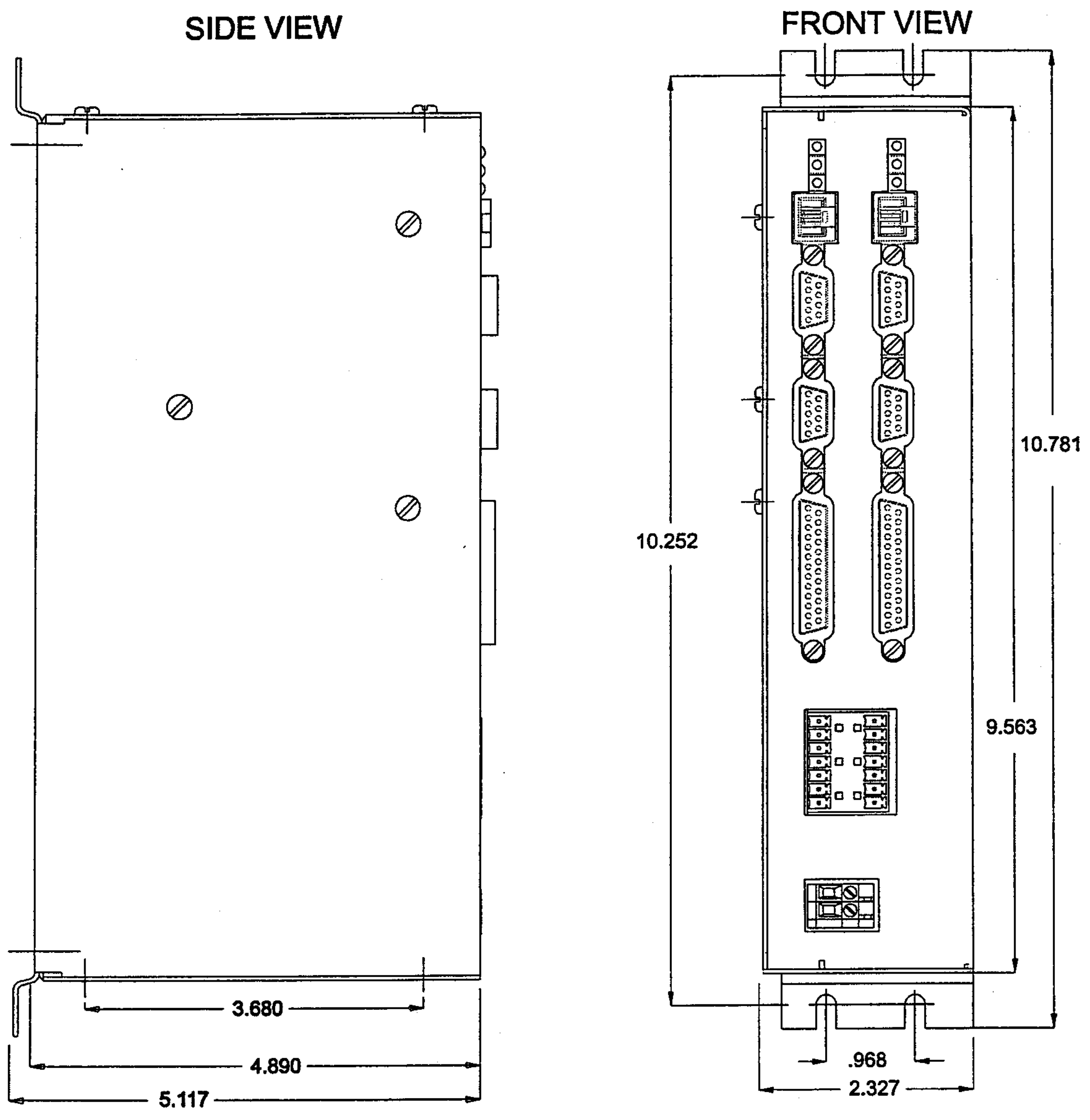


Figure 3.1
Mechanical Outline Drawing

3.2 - Mechanical Mounting of the Unit

Figure 3.1, Mechanical Outline Drawing, provides overall and mounting dimensions for the SS2000PCi-2. The unit should be solidly mounted within a control enclosure approved for the particular application. It is important to select a mounting location which will meet the environmental specifications listed in Section 4.1 Mechanical and Environmental Specifications. Avoid locations that expose the unit to extremes of temperature, humidity, dirt/dust, or vibration.

At least 2 inches of space must be left on the sides, top, and bottom of the unit to allow proper air flow for cooling of the unit.

Care must also be taken to allow proper and safe access to all wiring. It is best to avoid areas with high electrical noise. As discussed in Section 3.3 General Wiring Guidelines, this will help prevent incorrect operation due to electromagnetic interference.

3.3 - General Wiring Guidelines

Warning

Dangerous voltages, currents, temperatures, and energy levels exist within this unit, on certain accessible terminals, and at the motor. NEVER operate the unit with its protective cover removed! Caution should be exercised when installing and applying this product. Only qualified personnel should attempt to install and/or operate this product. It is essential that proper electrical practices, applicable electrical codes and the contents of this manual be followed strictly.

Superior Electric SLO-SYN controls and drives use modern solid-state digital electronics to provide the features needed for advanced motion control applications. Although care has been taken to ensure proper operation under a wide range of conditions, some user equipment may produce considerable electromagnetic interference (EMI) which can cause inappropriate operation of the digital logic used in the control, drive, or other computer-type equipment in the user's system.

In general, any equipment that causes arcs or sparks or that switches voltage or current at high frequencies can cause interference. In addition, ac utility lines are often "polluted" with electrical noise from sources outside a user's control (such as equipment in the factory next door). Some of the more common causes of electrical interference are:

- power from the utility ac line
- relays, contactors and solenoids
- light dimmers
- arc welders
- motors and motor starters
- induction heaters
- radio controls or transmitters
- switch-mode power supplies
- computer-based equipment
- high frequency lighting equipment
- dc servo and stepper motors and drives

The following wiring practices should be used to reduce noise interference.

Solid grounding of the system is essential. Be sure that there is a solid connection to the ac system protective earth ground (PE). Insure that there is a good electrical connection through the controller case to the control system enclosure . A separate grounding strap may be required to properly ground the unit to the control system enclosure. This strap should ideally be constructed using copper braid at least 0.5" in width. Use a single-point grounding system for all related components of the system (a "hub and spokes" arrangement). Keep the ground connection short and direct. Grounding through both a mechanical connection to the control enclosure and through a grounding strap is optimal.

Keep power and signal wiring separated. Power wiring includes ac wiring, motor wires, etc. Signal wiring includes inputs and outputs (I/O), encoder wiring, serial communications (RS232 lines), etc. If possible, use separate conduit or ducts for each. If the wires must cross, they should do so at right angles to minimize coupling.

Use separately bundled, shielded, twisted-pair cables for the drive to motor, encoder, serial communications, analog input, and digital I/O wiring. For other connections it is recommended that the shields be terminated at the Slo-Syn unit as well. Shield connections are provided on the unit terminal connectors for this purpose. All cable shielding should be terminated at ONE END ONLY. Grounding the serial communications connections at the opposite end from the controller may be necessary in some systems. If the cable shield must be connected at the opposite end from the Slo-Syn unit, the shield should NOT also be connected at the unit as this may cause a "ground loop" and introduce electrical noise problems.

Suppress all relays as close to the coil as possible to prevent noise generation. Typical suppressors are diodes, capacitors or MOV's. (See manufacturer's literature for complete information). Whenever possible, use solid-state relays instead of mechanical contact types to minimize noise generation.

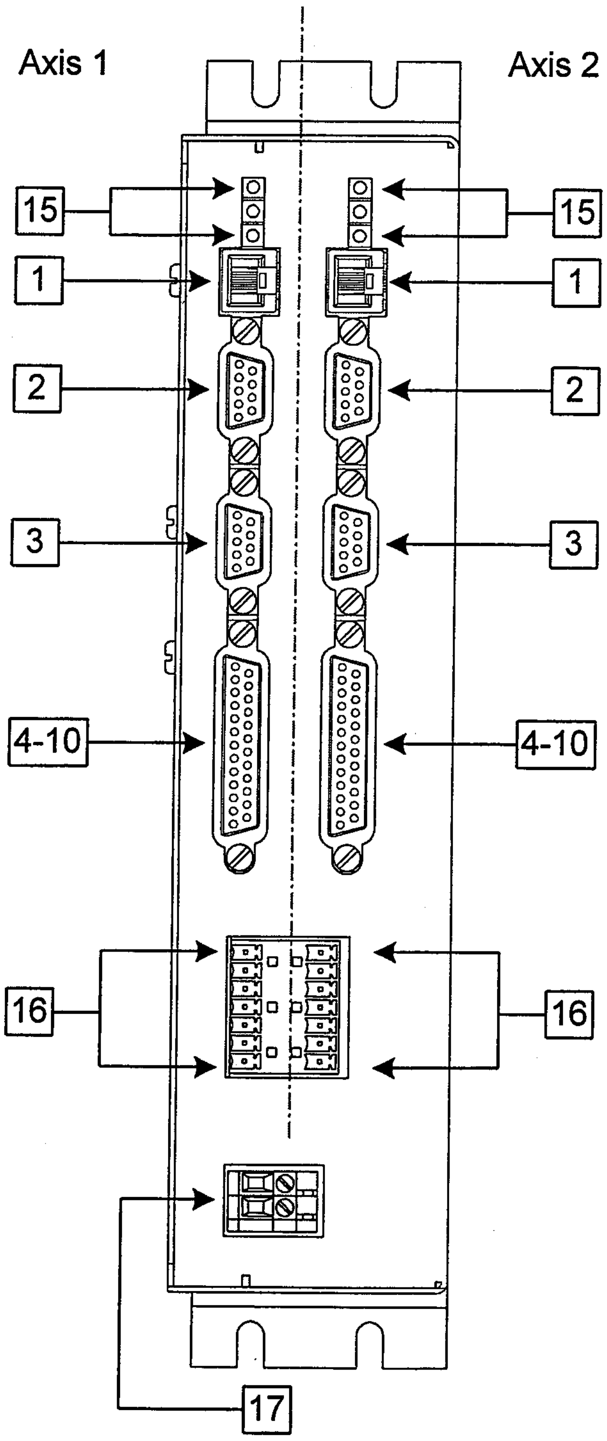
In some extreme cases of interference, it may be necessary to add external filtering to the ac line(s) feeding affected equipment, or to use isolation transformers to supply their ac power.

NOTE: Superior Electric makes a wide range of ac power line conditioners that can help solve electrical interference problems. Contact 1-860-787-3532 for further assistance.

3.4 - Hardware Connection Descriptions

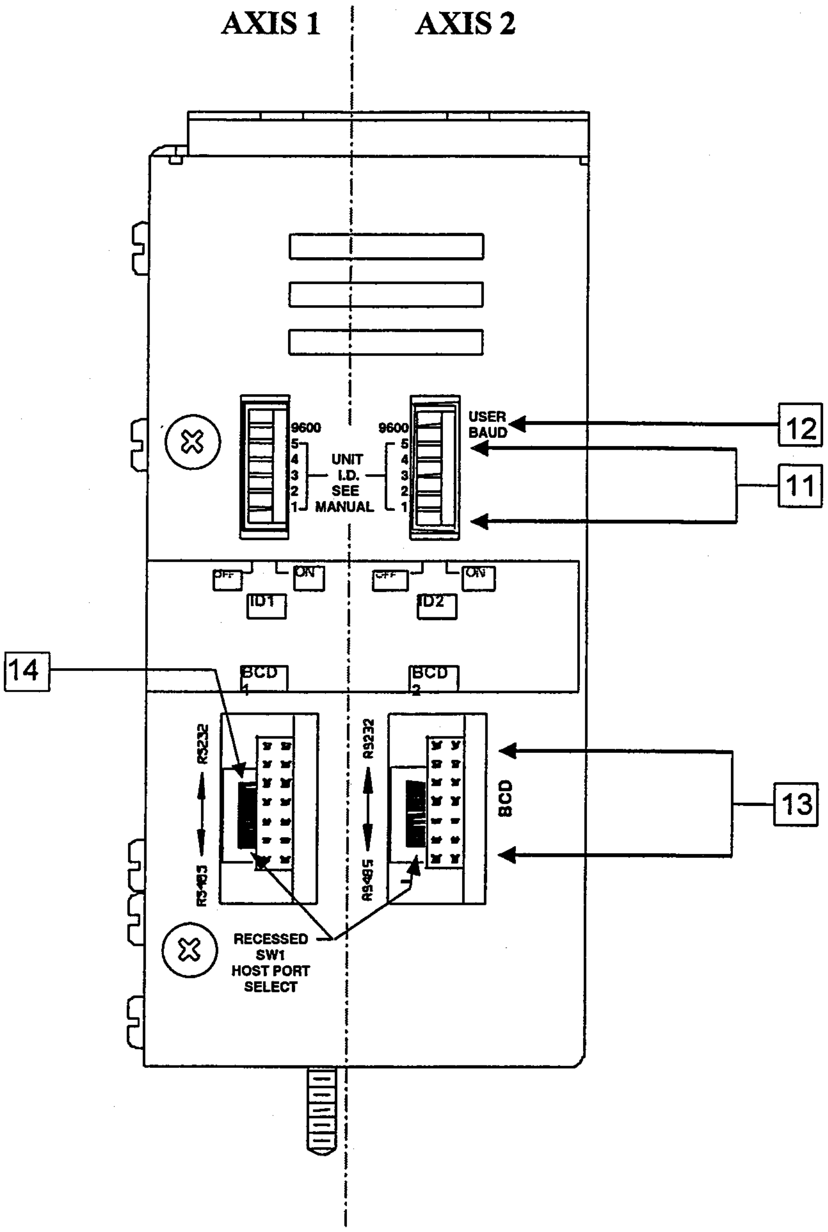
The following figures indicate the front and top views of the SS2000PCi-2 controller. The numbers in the boxes show the position of the various hardware connections to the unit Connections on the left are for Axis 1. Axis 2 connections are on the right. Use the index number in the boxes to find the description of each connection following the diagrams. The descriptions given here should provide a reasonable understanding of the nature of ~~each~~ signal and the way it should be wired into your system. More detailed technical information is available in Section 4.0 Hardware Specifications.

FRONT VIEW



TOP VIEW

All callouts are applicable to both axes



The following are brief descriptions of each connection to each of the 2 indexers in the SS2000PCi-2 unit. More detailed wiring diagrams are provided in Section 3.5

1 Serial Port 2

Auxiliary Serial Communications: Port 2

TX2+:	Transmit+ for Serial Port 2 (RS485)
TX2-:	Transmit- for Serial Port 2 (RS485)
RX2-:	Receive- for Serial Port 2 (RS485)
RX2+:	Receive+ for Serial Port 2 (RS485)

2 Serial Port 1

Host Serial Communications: Port 1

TX1-:	Transmit- for Serial Port 1 (RS485)
TX1+:	Transmit+ for Serial Port 1 (RS485 / RS232)
RX1+:	Receive+ for Serial Port 1 (RS485 / RS232)
RX1-:	Receive- for Serial Port 1 (RS485)
Serial GND:	Signal ground for Serial Port 1
+5V:	DO NOT USE AT THIS TIME.

3 Encoder

Encoder inputs for a closed loop stepper can be single-ended or differential phase quadrature.

B1+:	Encoder Channel B+ input
B1-:	Encoder Channel B- input.
A1+:	Encoder Channel A+ input.
A1-:	Encoder Channel A- input.
Z1+:	Encoder INDEX Channel Z+ input.
Z1-:	Encoder INDEX Channel Z- input.
+5V:	+5V supply for encoder.
GND:	Ground for encoder.

4 Inputs 1-8 (Isolated)

EVENT 1/ IN1; EVENT 2 / IN 2

These inputs can be used as mark registration and/or home inputs. If the inputs are not used for mark registration or home then the inputs can be used as programmable inputs. These inputs can be configured in the *Project Configuration & Setup.*

+LIMIT / IN3; -LIMIT / IN4

The +LIMIT or the -LIMIT may be used as inputs for limit switches or sensors. If limit switches are not needed, the inputs can be configured in the *Project Configuration and Setup* as programmable inputs.

RUN / IN5

The run input will start execution of the program. If auto-start is selected the program will start upon power up or RESET. RUN will also re-start a program if a **CLEAR** has been activated, or resume a program if a **FEEDHOLD** has been activated. If the **RUN** input is not needed the input can be used for a programmable input. This is done in the *Project Configuration & Setup.*

CLEAR / IN6

If the **CLEAR** input is open, the program or motion will stop. This input must be closed to run the program or start motion. If the **CLEAR** input is not needed the input can be used for a programmable input. This input can be configured in the *Project Configuration & Setup.*

FEEDHOLD / IN7

Activation of this input will cause motion to come to a **controlled stop**. After release of the **FEEDHOLD** input, activation of the **RUN** input will continue the program from the point the **FEEDHOLD** was activated. If the **FEEDHOLD** input is not needed it can be used as a programmable input. This input can be configured in the *Project Configuration & Setup.*

IN 8

This input can be used as a programmable input.

5 Outputs 1-4 (Isolated)

OUT 1, OUT 2, OUT3, OUT 4

These outputs can be used as programmable outputs.

6 Analog Input

The analog input connection allows a voltage from 0 VDC to +10VDC to be read into the unit.

ANALOG IN:	analog input.
GND:	Ground for analog input.

7 OPTO

+VOPTO; -VOPTO

A power supply for the optical isolators is **REQUIRED** *for proper I/O operation. This supply must be connected to the +VOPTO and -VOPTO pins.* The +12VDC and +12V COM power supply is available. This supply **MUST** be connected to +Vopto and -Vopto unless the user is to supply power for the I/O from a different source.

8 12 VDC Power Supply

12 VDC is available to power I/O . It is recommended that this power be connected to the +Vopto and -Vopto inputs on the controller as the discrete I/O supply.

+12V:	+12VDC output.
12V COM:	Common for the 12VDC supply.

The +12VDC supply current is limited to 100 mA. See Section 3.5.2 for connection of sink or source I/O.

9 IN COMMON

This input determines the current source of Inputs 1-8. If it is connected to +Vopto the inputs are set to the sinking mode. If it is connected to -Vopto the inputs are set to the sourcing mode.

10 GND

Signal Common for the **Analog In** signals. **This GND is not connected to 12VCOM or IN COMMON.**

11 Device ID Number Switch

The DIP switches will allow up to 32 devices to be daisy chained together. Each unit must have a unique ID number per the table of ID settings in Section 3.2, Baud rate and Unit ID Switches.

12 Baud Rate Switch

This switch is read only at power-up or after a reset command. In the **off** position the baud rate is forced to 9600. In the **on** position the baud rate for the loaded project is used. The User Baud rate is selected in the project *Configuration and Setup*. If no user program is loaded the default, 9600, baud rate is used. If the baud rate in the configuration and setup is not known, use 9600 at power-up.

13 BCD Port

BCD Port / I/O

This port can be used as either a BCD port, consisting of 7 numbers and a sign **(Superior Electric Part # 221157-002, includes BCD switch and 18" ribbon cable)**, or used for additional outputs and inputs*.

BCD0/IN9 : BCD switch data 0 or program input 9.

BCD1/IN10: BCD switch data 1 or program input 10.

BCD2/IN11: BCD switch data 2 or program input 11.

BCD3/IN12: BCD switch data 3 or program input 12.

BCD4/IN13: BCD switch data 4 or program input 13.

BCD5/IN14: BCD switch data 5 or program input 14.

BCD6/IN15: BCD switch data 6 or program input 15.

BCD7/IN16: BCD switch data 7 or program input 16.

BCD STR0/OUT5: BCD switch Strobe 0 or output 5

BCD STR1/OUT6: BCD switch Strobe 1 or output 6

BCD STR2/OUT7: BCD switch Strobe 2 or output 7

BCD STR3/OUT8: BCD switch Strobe 3 or output 8

GND: Signal Common for Inputs and outputs

*__Note: When the BCD port is used for additional I/O, all inputs are non-isolated, and all outputs are open-collector (7406) active low.__

14 Serial Port 1 (RS232 / RS485) Switch

This switch allows Serial Port 1 to be configured for RS232 or RS485 4-wire communications.

Caution

Use care when accessing this recessed switch; do not damage adjacent components when changing its position.

15 LED's

These LED's show conditions that may be occurring in the controller.

POWER:	The power LED indicates that there is AC power applied to the drive and that the logic supply is active.
BUSY:	Signifies that motion is occurring on the motor.
FAULT:	Indicates that an error has occurred in the controller.

16 Drive Connections

- **OPTO** — +5VDC output power for connection to drive opto coupler supply.
- **PULSE** — Open collector output that supplies pulses to the drive. Drive should count ON-OFF transitions (low to high)
- **DIR** — Open collector output that signals drive which direction to turn the motor.
- **AWO** — Open collector output that turns off drive when on (low).
- **REDUCE** — Open collector output that signals drive to reduce motor current when active (low).
- **BOOST** — Open collector output that signals drive to boost motor current when active (low).
- **READY** — Logic input signal indicating that the drive is ready to go. This signal is active when at +5V DC (OPTO power).

17 AC Power

AC CONNECTIONS

These inputs are for connection of single phase AC power. The input power range is from 95VAC to 265VAC, 0.4 Amps, 50/60 HZ.

18 Chassis Ground

Grounding locations for the motor and AC connections. It is critical that a solid connection from Protective Earth Ground be connected to the chassis ground. The Ground wire must be at least as large as the AC supply power wiring.

3.5 - Wiring Diagrams

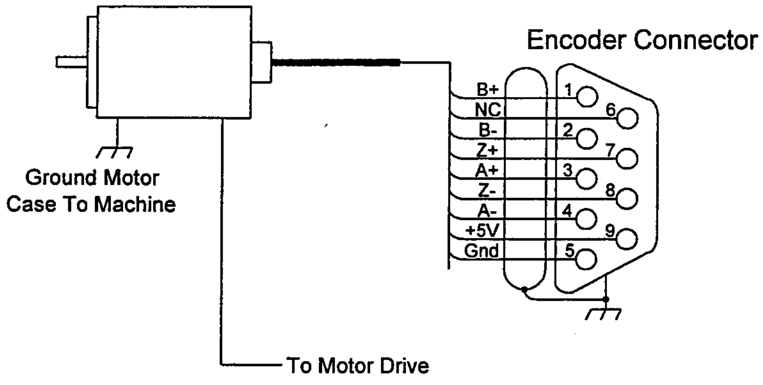
Caution Warning

This section provides wiring diagrams for each connection. Remember to follow the General Wiring Guide outlined previously.

NEVER wire the unit with the power on! Serious injury as well as damage to the unit may result.

3.5.1 - Encoder Connections

The encoder connections are only required for a closed loop stepper drive and the connection scheme is depicted below. **Note:** It is **IMPORTANT** that the encoder and motor cables be shielded and that the shields be connected to their appropriate connector terminals. Single ended TTL or open collector encoder signals must be connected to the "+" terminal of each channel.

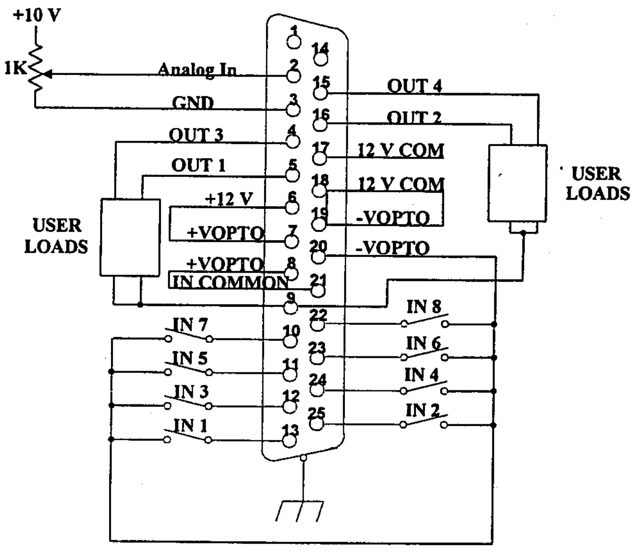


3.5.2 - Input / Output Connections

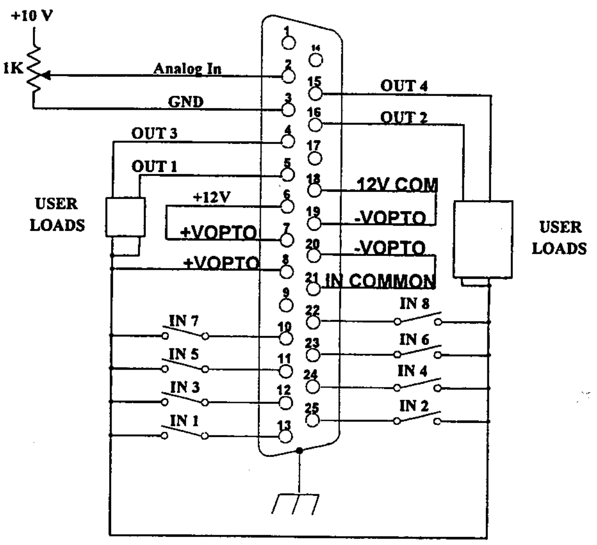
The I/O connections consist of; 8 general purpose inputs, 4 general purpose outputs, and 1 analog input. The 8 general purpose input signals can be sinking or sourcing opto-isolated inputs. The input mode (sink or source) applies to all 8 inputs. They may not be individually selected as sink or source. The 4 general purpose output signals are sinking only opto-isolated outputs. The analog input has a voltage range from 0 to +10 volts.

I/O Connection examples using the +12 Vdc internal power supply are depicted below. The general purpose input connections are shown for both sinking and sourcing inputs.

Internal Power Supply
Inputs Sinking – Outputs Sinking

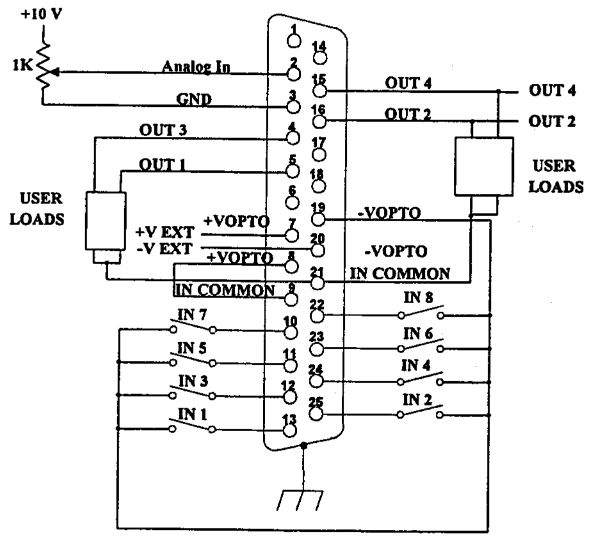


Internal Power Supply
Inputs Sourcing – Outputs Sinking

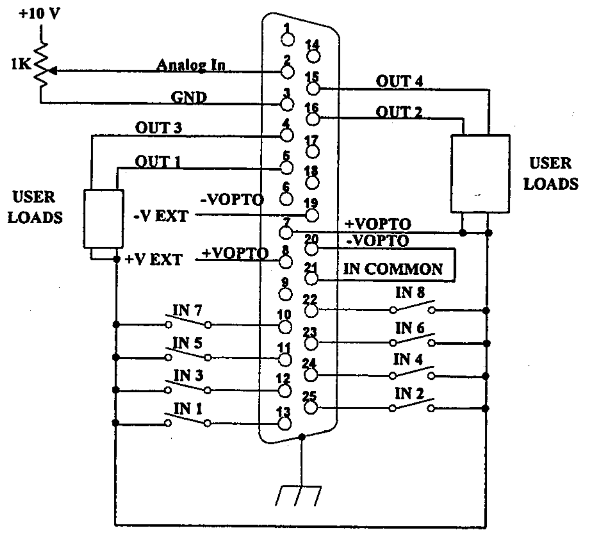


Depicted below are the I/O connections when using an External Power supply. Use these connections if you are not using the +12 Vdc internal power supply. The general purpose input connections are shown for both sinking and sourcing inputs.

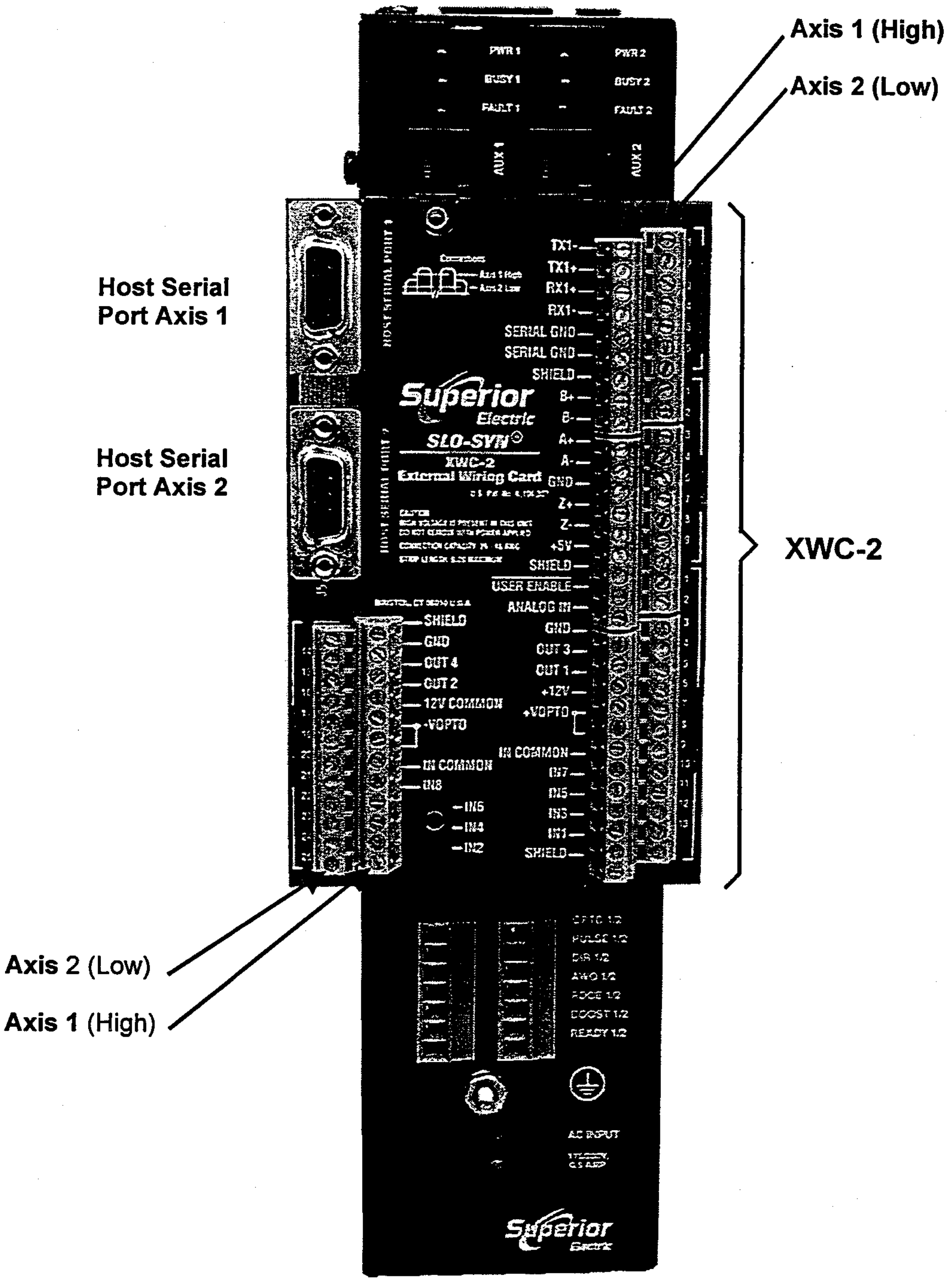
External Power Supply
Inputs Sinking – Outputs Sinking



External Power Supply
Inputs Sourcing – Outputs Sinking

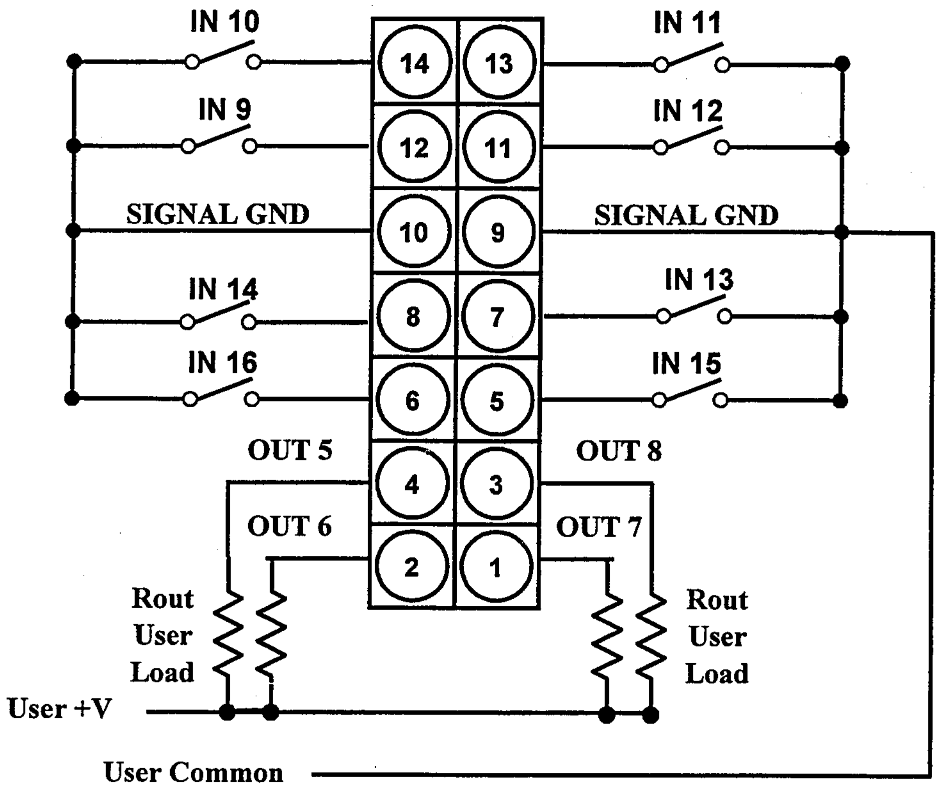


An optional external wiring card is available which provides individual terminals for each I/O point on the "D" style connectors utilized on the unit. The External Wiring Card mounts over the D connectors on the face of the unit. All connections are brought out to individual clamp down type terminal connections. The pinout descriptions for the terminal blocks are identical to those for the D connectors described in this section. The inner most terminal blocks (higher profile) are used for Axis 1 connections. The outer (lower profile) terminal blocks are used for Axis 2. If you would like to utilize the optional external wiring card, contact customer service or your distributor and request Part Number XWC-2. The optional external wiring card is shown below. Note: The USER ENABLE signal is not used when the XWC-2 is used with the SS2000PCi-2.



The connections for the BCD I/O connector when used as general purpose signals are depicted below.

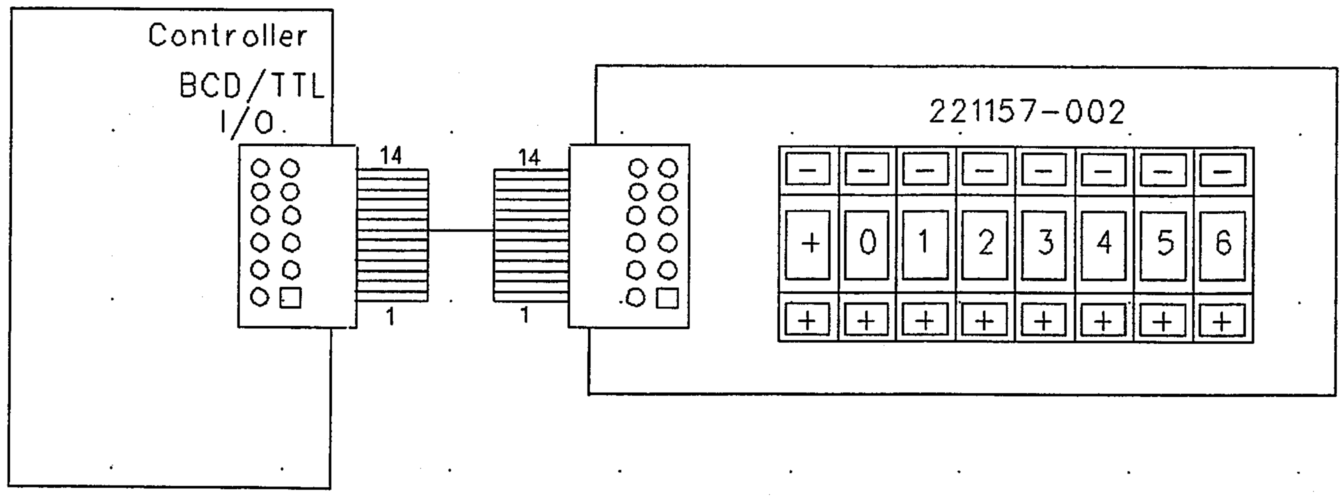
Note: These inputs and outputs are not isolated.



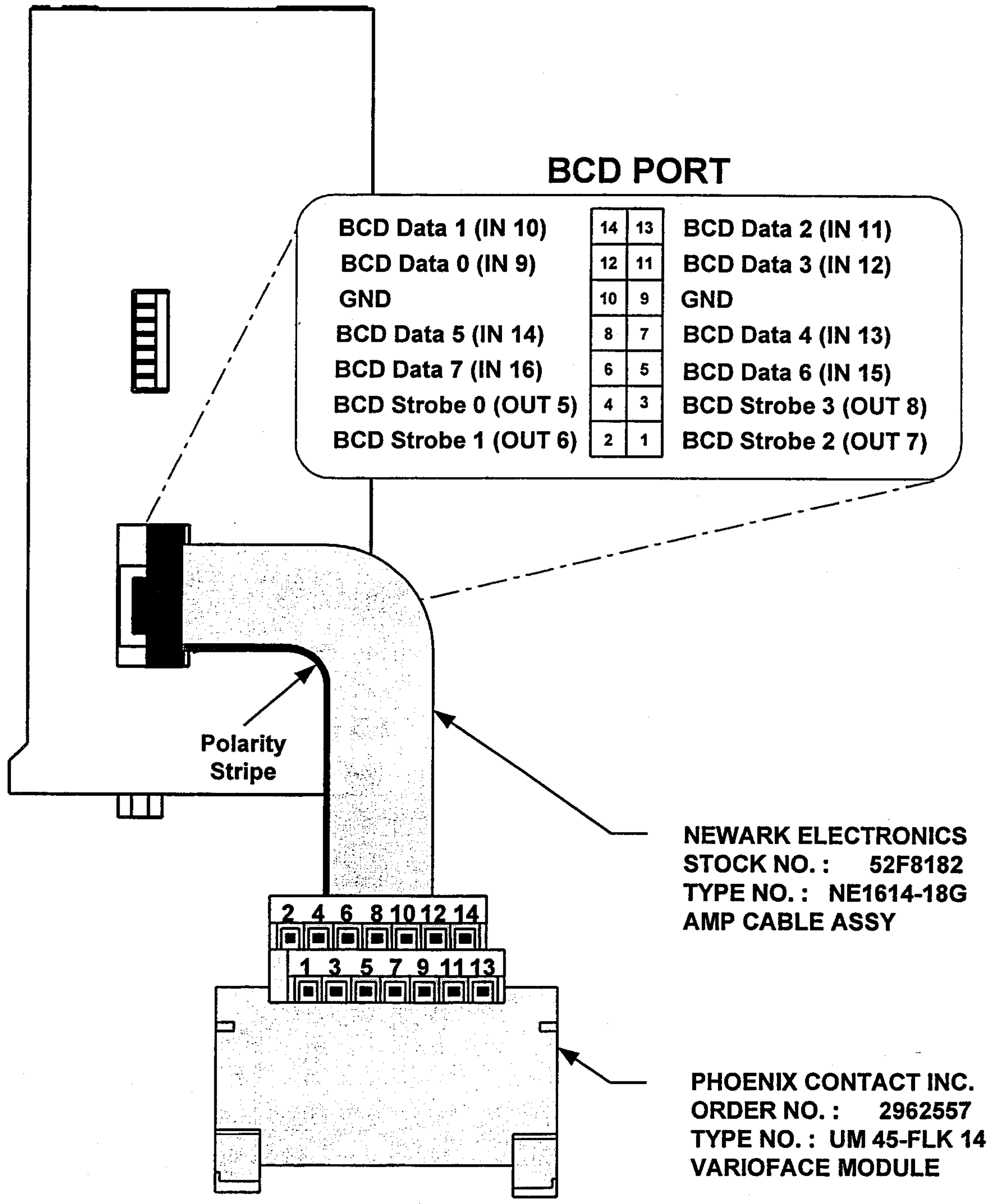
Recommended Output Loads

User + V	Rout
5 VDC	500 ohm
12 – 15 VDC	1.5 Kohm
24 VDC	2.5 Kohm

The connections for the BCD / TTL I/O connector when used as a BCD port are depicted below.
Note: The Superior Electric BCD switch interface P/N 221157-002 is shown.

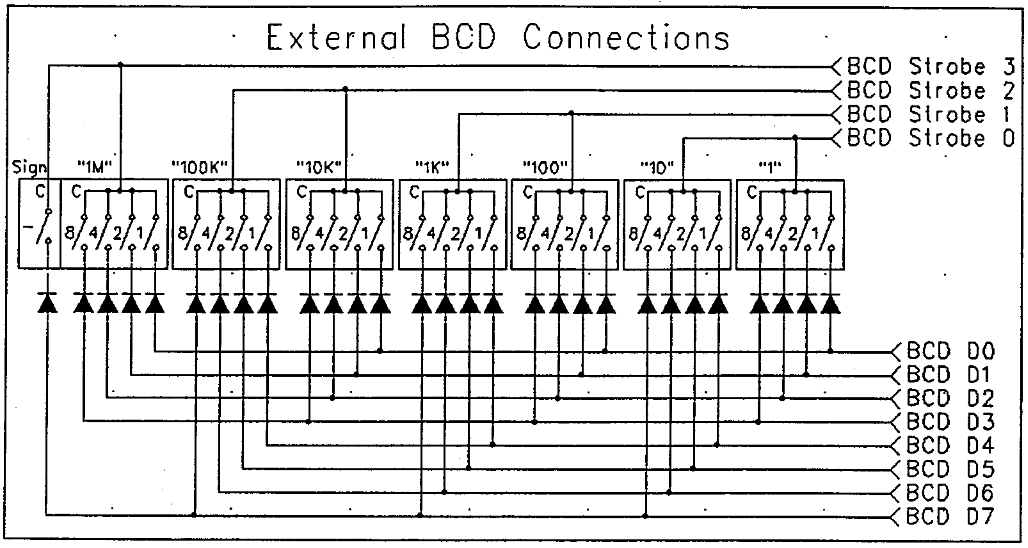


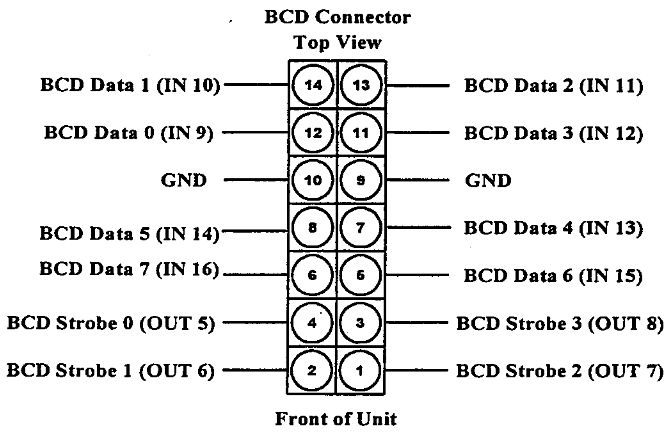
RECOMMENDED CONNECTION FOR USING BCD PORT AS NON-ISOLATED I/O



The connections for the BCD / TTL I/O connector when used as a BCD port are depicted below.

Note: An External BCD Configuration is shown below.

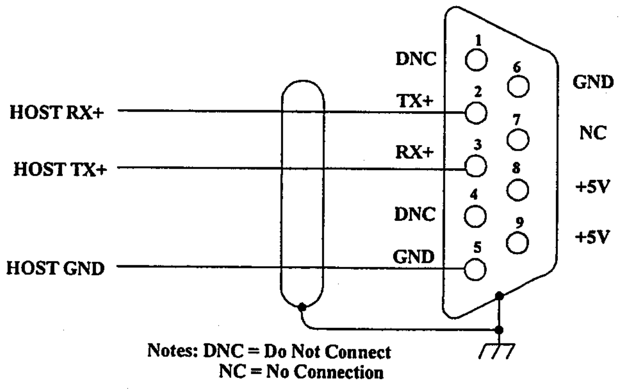




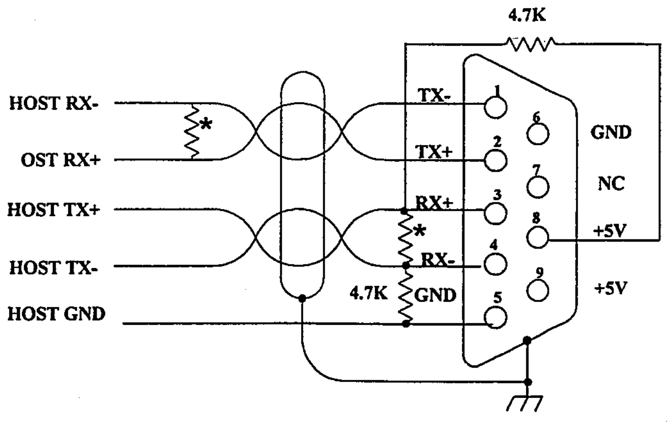
3.5.3 RS232 / RS485 Host Serial Communications Connections

This serial port is used for communications and programming of the controller from a personal computer (PC). The port can be configured for RS232 or RS485 operation. A slide switch has been provided for making this selection. The factory default is RS232. The connection diagram for both modes has been provided, See Section 3.4 Top View. **Note: When wired for RS485 operation, a cable with individual twisted pair wires must be used.** The termination resistors indicated with an * must have a value of 120 ohms. The pull up/down resistors have a value of 4.7kΩ. If multiple units reside on the RS485 bus only ONE set of three resistors should be used at the end of the transmission line bus. The resistor connected to the TX+ and TX- signals may be required if the terminal device does not provide the termination resistor internally. If the terminal device does provide the resistors internally, the resistors connected to the TX+ and TX- are not required.

Host (RS232 Selected)



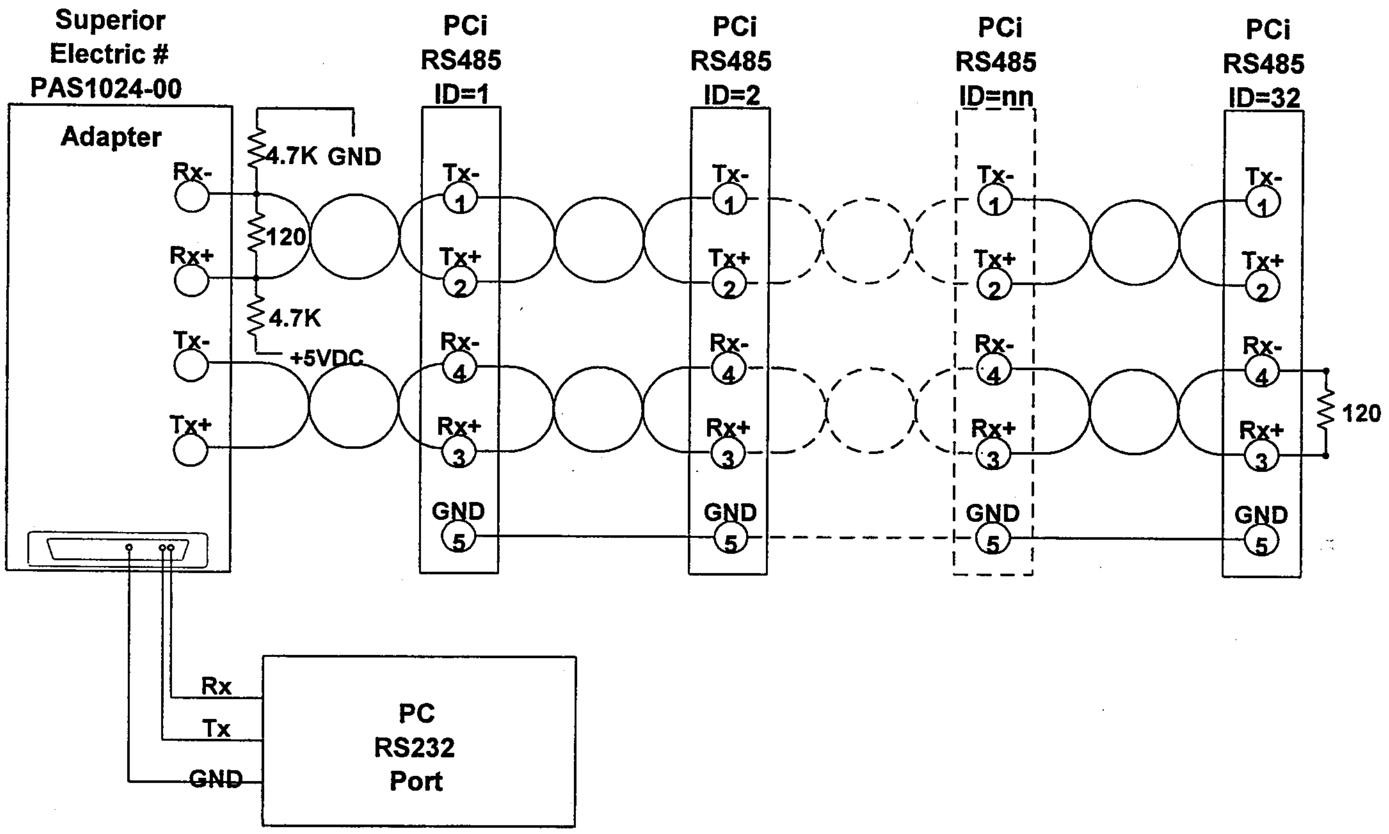
Host (RS485 Selected)



3.5.3.1 - RS485 Host Daisy Chaining Connections

Connection in a daisy chain configuration requires that the Host port of all units be wired as RS-485. Each unit must also be switched to RS485 Host communications mode by moving the recessed slide switch into the RS485 position. The switch is accessible through the removable portion of the label on top of the unit near the BCD I/O port. Be sure that the unit is off when changing the switch position.

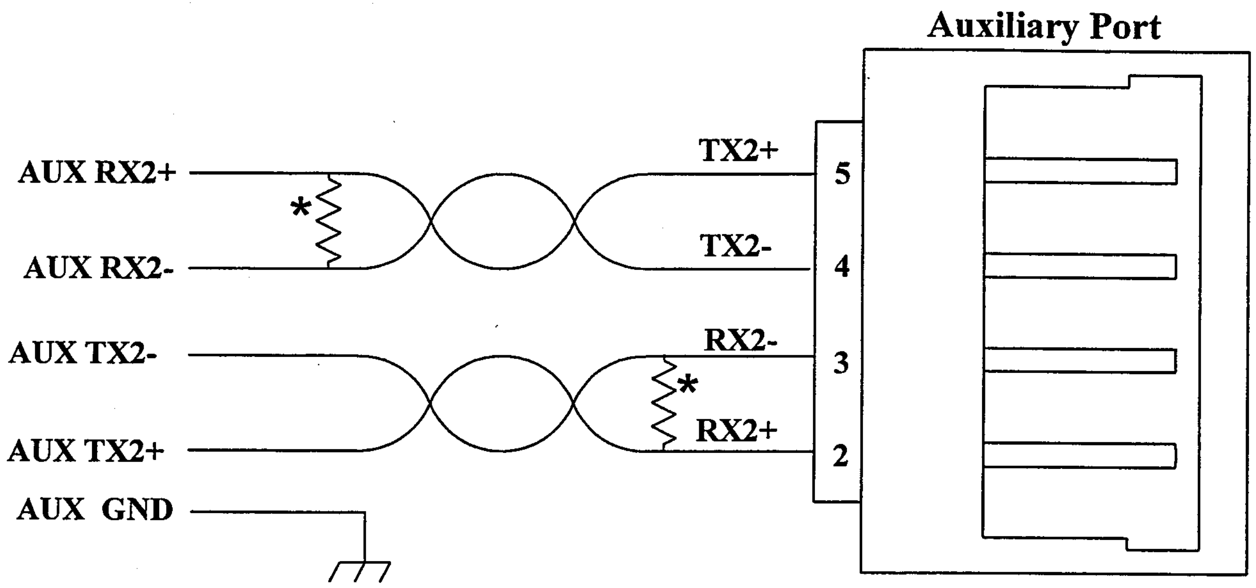
Important!! Connection to a PC that has an RS-232 port only can be accomplished by using an RS-232 to RS-485 four wire adapter such as Superior Electric part number PAS1024-00 as shown. If your PC has an RS-485 port, the adapter is not required.



3.5.4 RS485 Auxiliary Communication Connections

The auxiliary serial port is used for serial communications to and from other devices, such as PLC's or operator interface panels. This serial port is RS485 only and uses a telephone jack for the connections. The wiring connection diagram is shown below. **Note: A cable with individual twisted pair wires should be used.** The termination resistors indicated with an * must have a value of 120 ohms.

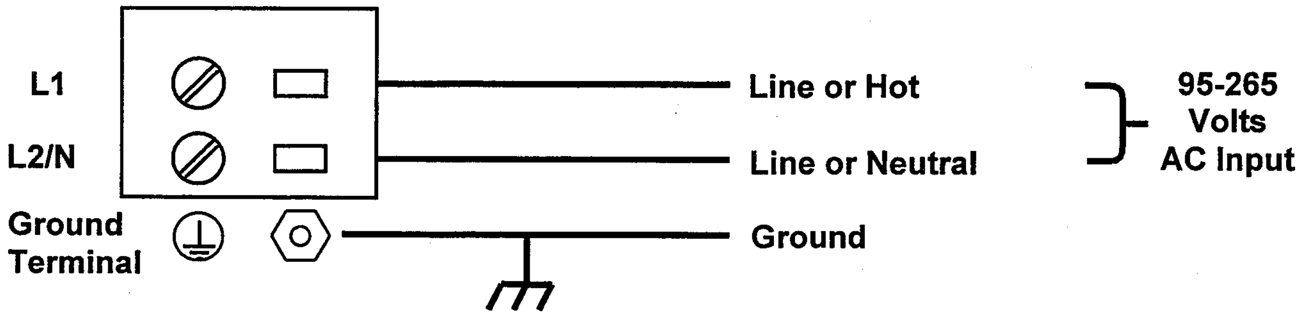
If multiple units reside on the RS485 bus, ONE resistor should be used at the end of the transmission ~~line~~ bus. The resistor across the TX+ and TX- signals may be required if the terminal device does not provide a termination resistor internally. If the terminal device does provide the resistor internally, the resistor across TX+ and TX- is not required.



3.5.5 - AC Power Connections to the Unit

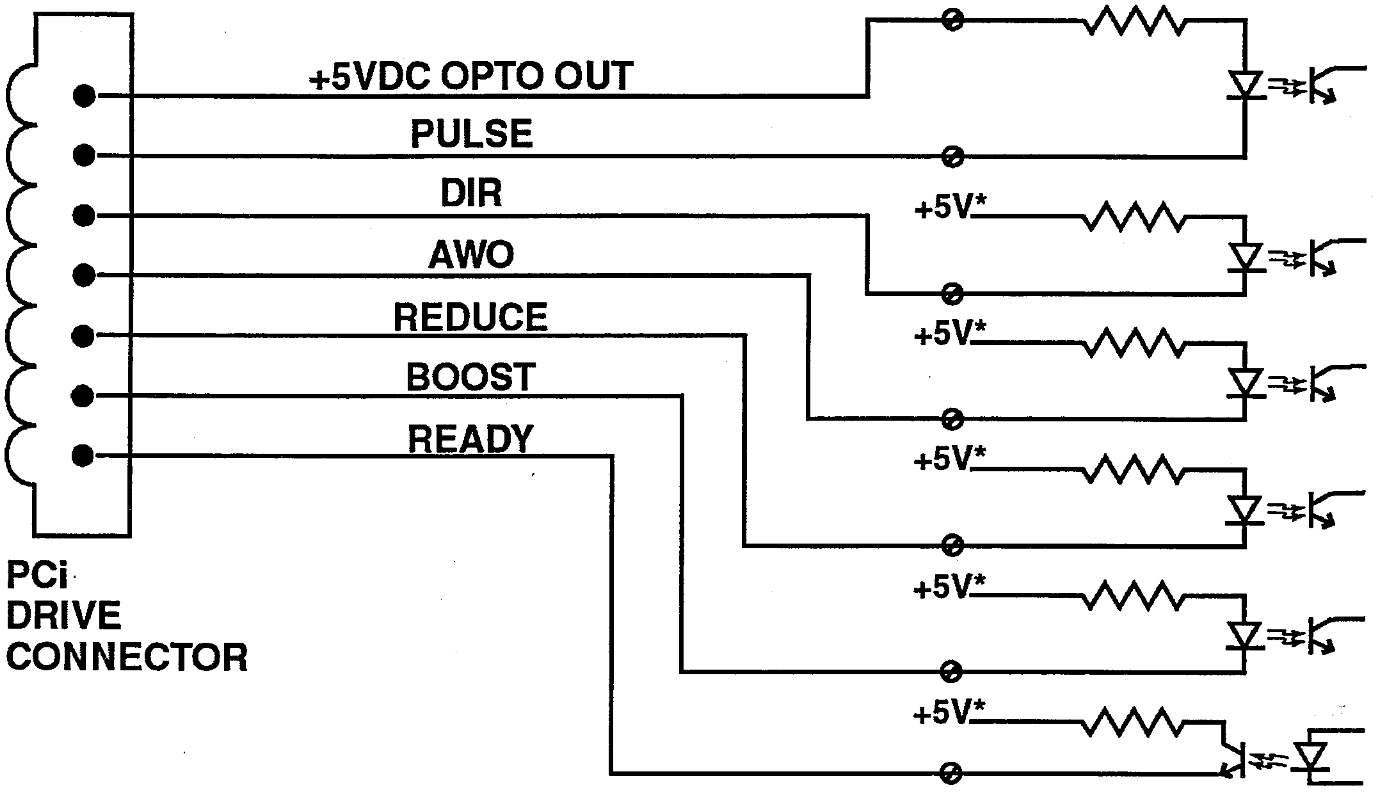
Connect the two (2) AC IN terminals to the input AC line. The line voltage can be from 95 VAC to 265 VAC 50/60 Hz.

Do not exceed the voltage rating of the drive and motor. Damage may occur if the ratings are not observed.



3.5.6 Drive Connections

A typical optically isolated connection to a stepper drive is shown below:



* Note that on some drives the connection from the top side of the opto couplers is made internally. Other drivers may have two terminals per I/O point (uncommitted opto couplers). These drives would require external connection from the high side of the opto coupler to +5VDC opto out. A current limiting resistor may also be required. Consult your drive manual and verify proper connections prior to energizing your system.

4.1 Mechanical and Environmental Specifications

Size: (Inches)	2.34W x 9.56H x 5.12D
(mm)	59.44W x 242.82H x 130.05D
Weight	2.88 lbs (1.31 kg)
Operating temperature:	+32° F to +122° F (0° C to +50° C)
Storage temperature:	-40° F to +167° F (-40° C to +75° C)
Humidity:	95% maximum, non-condensing
Altitude:	6,560 feet (2000 meters) maximum

4.2 Electrical Specifications

AC Input Range	95 to 265 VAC, 50/60 Hz
AC Current	.4 amperes
Fuse Rating**	250 volts, 2 amperes
Fuse Type**	Type 3AG

** If this fuse blows, the power supply will be prevented from energizing any of its outputs, hence, the unit will not operate. Usually, this fuse will only blow if an internal failure occurs. In order to ensure safety the specified rating and type of fuse **MUST BE USED.**

The following specifications are the same for each of the two indexers/axes.

4.2.1 Isolated Digital I/O

12 VDC I/O Power:	11.5 to 14 VDC @ 100 mA
Inputs (IN1 – IN8):	
Sink mode: (IN COMMON tied to +Vopto)	
On state voltage range (+Vopto =12V with -Vopto = 0V):	0V to 6VDC
Input Current; (VIN = 0V), +Vopto=12V, -Vopto=0V:	-6mA
Source mode: (IN COMMON tied to –Vopto = 12V common)	
On state voltage range with -Vopto = 0V:	4.5V to 24VDC
Input Current; (VIN=12V) with -Vopto=0V :	6mA
Response time (sink or source):	
Opto turn on delay:	10µS typical
Opto turn off delay:	75µS typical
User OPTO coupler power supply range (if not using the internal +12 V supply):	5-24 VDC
Programmable Outputs (OUT1 – OUT4):	
Sink mode only:	
Continuous Current rating:	250 mA max
Maximum collector voltage with -Vopto = 0V:	25V
On state voltage @ 250mA:	1.5V max

5.1 - PROGRAMMING

5.1.1 - General Description Of Programming

Programming of any sort requires planning and forethought. Programming your Controller is no exception. This section provides aids to facilitate your planning process.

5.1.1.1 - What is Programming?

A program is a list of discrete lines or command strings that, taken together in sequence, provide the information needed to get a machine to perform your predetermined sequence of instructions. These instructions can, in the case of Programmable Motion Controllers, cause the motor to move at certain speeds for given distances, read various inputs or set outputs, all used to accomplish different machine-related tasks.

5.1.1.2 - What's in a Program?

A program consists of many individual lines organized in a prescribed sequence. The **SS2000PCi-2** system uses an English language, BASIC-type computer programming language (SEBASIC). This makes it easy and intuitive to write and read machine control programs. SEBASIC supports many higher level language features, such as statement labels, subroutines, for-next and do-while loops for program flow control making it easy to write concise, well organized, easily debugged programs. Also, there are built in mathematics, Boolean functions and two dimensional array capability. Finally, the motion, I/O, and timing commands are easy to understand, remember and apply.

In addition to lines of program code , the controller uses and saves a series of set-up parameters. These parameters are set by the user in the Configuration & Setup section of the project.

5.1.1.3 - How is the Controller Programmed?

The programming environment called **Motion Controller Programming Interface** is supplied on a diskette. This software provides an easy to use environment for developing a user project. Detailed instructions on how to install this software on your PC are provided in this manual. Since each axis operates independent of the other they both have to be programmed if you want to utilize both. Section 7 provides details how to program and wire the two axes to "work together" one at a time.

5.1.2 -What are Host Commands?

Host commands go straight from your input device (PC or terminal) to the controller. They allow parameters to be set or interrogated, motion to be started or stopped, and program execution to be started or stopped, etc..

5.1.3 - Memory Types and Usage

The controller uses two kinds of memory, volatile and non-volatile. RAM (Random Access Memory) is called **Volatile Memory** because when power is removed from the controller, all of the information in that memory is lost. The Controller stores the program variables in RAM.

The second kind of memory is **Non-Volatile Memory**, or FLASH memory. The information stored in this type of memory is not lost when power is removed. FLASH memory is used by the controller for storing the Operating System as well as the User Program.

A Controller program can have hundreds of lines of code. Code is simply an organized listing of program commands. Because of the wide variety of program commands it is impossible to state how many lines can be stored in the controller. The amount of free memory remaining can be obtained with the FREEMEM host command.

5.1.4 - How to organize your Project

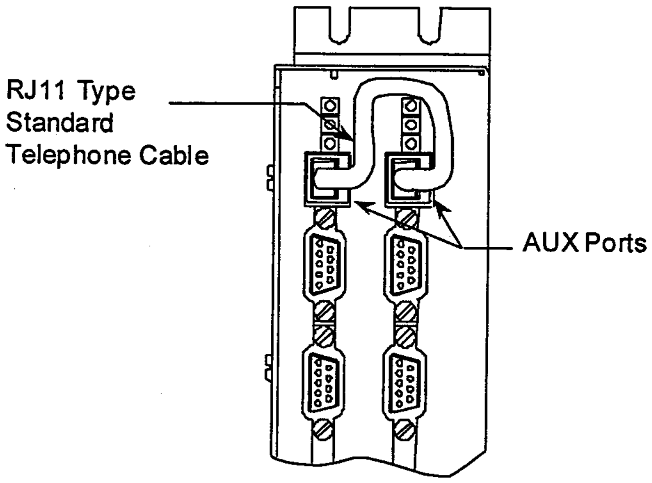
A project consists of a Configuration & Setup section and the user program. The Configuration & Setup section allows access to project related parameters and conditions via folders. The user program performs your predetermined sequence of instructions.

A good program will consist of initialization, main program, Interrupt routines, Subroutines and Error Handler sections. The Interrupt routines, Subroutines and Error Handler sections are optional. A typical Program Development Block Diagram is provided in Figure 5.1.

5.1.4.1 - Initialization Section

Variable names and data types (Integer, Integer Array, Real and Real Array) are defined in this section. Also the condition's which will trigger the individual Interrupts (INTR1-INTR4) may be defined.

7.4. Two Independent Axis Control Programming Example (Using AUX Port)



The following is an example of how to program the two independent axes of the SS2000PCi-2 to work together using the Auxiliary ports to communicate using a standard telephone cable (user provided).

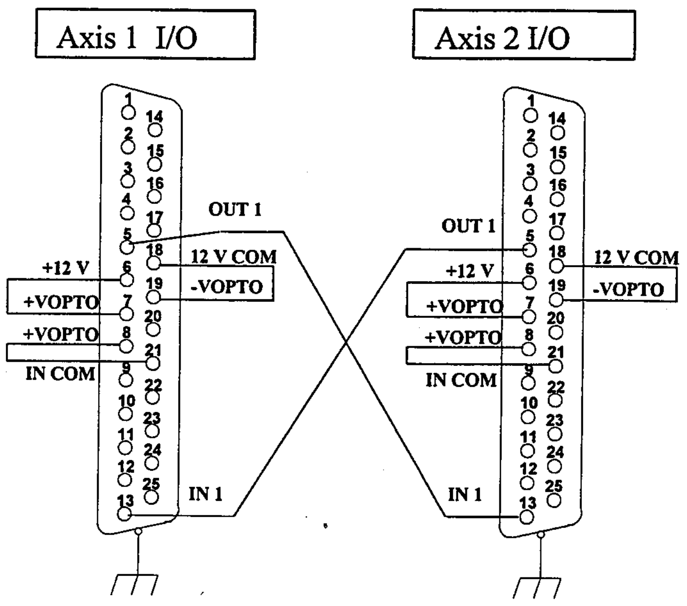
Program Code (Axis 1):

```
'****************** Sample program showing PCi-2 communicating using Auxiliary Ports. ******************
'*                                                                                                       *
'*      The Auxiliary Port uses RS485 operation mode and is indicated as Port 2 in each command.          *
'*      A standard RJ-11 cable can be used.  It is STRONGLY recommended that both ground screws           *
'*      be connected to the same ground point. A program must be written for  each PCi motion control.    *
'********************** Download this program into the unit designated as AXIS 1. ***********************
'Program Description:  Program auto start MUST be enabled in the I/O folder of the configuration. This program will
'move this axis clockwise 10 units, communicate to the second PCi that motion is complete, wait for the second PCi
'to transmit that motion is complete, evaluate whether to move forward or reverse based the STRING sent from
'the second PCi, and finally move clockwise or counter-clockwise 5 units.

Declaration:    'Declaring all variables is required.  An INTEGER is a whole number, i.e. 1,2,3;
                'A REAL can be a decimal number i.e., 1.543; A STRING is a character or a word.
INTEGER a, b, c
REAL x, y, z
STRING A$, B$, C$

Motion_PCi_Master:
   MOVEI=10                               'Commands a move of 10 units for this axis.
   WAITDONE
   PRINT#2, "k"; CHR$(13);                'Send a message using the Auxiliary Port to the other axis.
                                          'Note: CHR$(13) MUST be sent.
Controller_Pause:
   INPUT#2,A$                             'Holds the program cursor until a word is received from the Auxiliary Port
                                          'followed by a carriage return.
Data_Evaluated:                           'Evaluates the data received and proceeds to the next routine.
   IF A$="Go_Left" THEN GOTO Forward
   IF A$="Go_Right" THEN GOTO Reverse
   IF A$ <> "Go_Left" OR A$ <> "Go_Right" THEN GOTO Controller_Pause
Forward:                                  'Rotates the motor clockwise.
   MOVEI=5
   WAITDONE
END

Reverse:                                  'Rotates the motor counter-clockwise.
   MOVEI=-5
   WAITDONE
END
```

Program Code (Axis 2):

```
'************* Sample program showing PCi-2 communicating using Auxiliary Ports. ************************
'**************************************************************************************************
'****** The Auxiliary Port uses RS485 operation mode and is indicated as Port 2 in each command. **************
'****** A standard RJ-11 cable can be used.  It is STRONGLY recommended that both Ground screws ************
'****** be connected to the same ground point. A program must be written for each PCi motion control axis.  ******

'****** Download this program into the unit designated as AXIS 2.  **************************************

'Program Description:  Program auto start MUST be enabled in the I/O folder of the configuration.  This program will 'pause
until data is transmitted from the first PCi.  Upon receiving data, this axis will move clockwise.  When motion is
'complete the statement "Go_Left" will be sent to the first PCi.

Declaration:        'Declaring all variables is required.  An INTEGER is a whole number, i.e. 1,2,3;
                    'A REAL can be a decimal number i.e., 1.543; A STRING is a character or a word.
INTEGER a, b, c
REAL x, y, z
STRING A$, B$, C$

Controller_Pause:
        INPUT#2,A$                  'Holds the program cursor until a word is received from the Auxiliary Port
                                    'followed by a carriage return from the first PCi.
Motion_PCi_Master:
        MOVEI=2                     'Commands a move of 2 units for this axis.
        WAITDONE
        PRINT#2,"Go_Left";CHR$(13); 'Send a message using the Auxiliary Port to the first axis.
                                    'Note: CHR$(13) MUST be used following the semicolon.
END
```

7.5. Two Independent Axis Control Programming Example (Using I/O Port)



Program Code (Axis 1):

```
'******************* Sample program showing PCi-2 communicating using Inputs and Outputs ******************
'*****************************************************************************************************
'* Inputs and outputs can be used as indicators from the 1st PCi to the 2nd PCi.  One isolated input and one isolated output*
'* on each PCi MUST be dedicated to communicate completion of execution, i.e., motion. It is STRONGLY recommended*
'* that both ground screws be connected to the same ground point. A program must be written for  each PCi motion control.*
'*****************************************************************************************************
'****** Download this program into the unit designated as AXIS 1. *********************************
'Program Description:  Program auto start MUST be enabled in the I/O folder of the configuration. This program will
'move this axis clockwise 10 units, communicate to the second PCi that motion is complete by activating output #1.
'This PCi  will pause until input #1 is activated by the 2nd axis.  Once input #1 is on, this axis will move 5 units clockwise.
'The program will pause until input #1 is activated again, allowing motion of 5 units in the counter-clockwise direction.
Declaration:    'Declaring all variables is required.  An INTEGER is a whole number, i.e. 1,2,3;
                'A REAL can be a decimal number i.e., 1.543; A STRING is a character or a word.
INTEGER a, b, c
REAL x, y, z
STRING A$, B$, C$
Motion_PCi_Master:
        MOVEI=10                    'Commands a move of 10 units for this axis.
        WAITDONE
        OUT(1)=1                    'Activates output #1 connected to input #1 of the second PCi.
        WAIT=2                      'Time delay of 2 seconds.
        OUT(1)=0                    'Deactivates output #1.
Controller_Pause:
        DO : LOOP UNTIL IN(1)=1     'Holds the program cursor until input #1 is activated.  At a
                                    'true condition, this program continues to the next line.
Forward:                            'Rotates the motor clockwise.
        MOVEI=5
        WAITDONE
        DO : LOOP UNTIL IN(1)=1     'Holds the program cursor until input #1 is activated.  At a
                                    'true condition, this program continues to the next line.
Reverse:                            'Rotates the motor counter-clockwise.
        MOVEI=-5
        WAITDONE
END
```

Program Code (Axis 2):

```
'**************** Sample program showing PCi-2 communicating using Inputs and Outputs ***************
'*******************************************************************************************
'****** Inputs and outputs can be used as indicators from the first PCi to the second PCi.  One isolated input ****
'****** and one isolated output on each PCi MUST be dedicated to communicate completion of execution, *****
'******  i.e., motion.  It is STRONGLY recommended that both ground screws be connected to the same ******
'****** ground point. A program must be written for  each PCi motion control.  **************************

'****** Download this program into the unit designated as AXIS 2.

'Program Description:  Program auto start MUST be enabled in the I/O folder of the configuration. This program will
'wait until input #1 has been activated by the first PCi control.  A move of 10 units clockwise will be completed.
'Output #1 will be activated to indicate to the first PCi that motion has been completed.  This axis will move 5 more
'units and then wait until input #1 is activated.  If the condition becomes true, this axis will move 5 units counter-
'clockwise and end.

Declaration:        'Declaring all variables is required.  An INTEGER is a whole number, i.e. 1,2,3;
                    'A REAL can be a decimal number i.e., 1.543.
INTEGER a, b, c
REAL x, y, z


Controller_Pause:
        DO : LOOP UNTIL IN(1)=1         'Holds the program cursor until input #1 is activated.  At a
                                        'true condition, this program continues to the next line.
Motion_PCi_Second:
        MOVEI=10                        'Commands a move of 10 units for this axis.
        WAITDONE
        OUT(1)=1                        'Activates output #1 connected to input #1 of the second PCi.
        WAIT=2                          'Time delay of 2 seconds.
        OUT(1)=0                        'Deactivates output #1.

Forward:                                'Rotates the motor clockwise.
        MOVEI=5
        WAITDONE

        DO : LOOP UNTIL IN(1)=1         'Holds the program cursor until input #1 is activated.  At a
                                        'true condition, this program continues to next line.

Reverse:                                'Rotates the motor counter-clockwise.
        MOVEI=-5
        WAITDONE
END
```

Section 10

Host – Slave Control and Peer-to-Peer Control of Axis 1 & 2

10.1 Introduction to Host-Slave and Peer-to-Peer Control

This section of the SS2000-PCi-2 manual describes the use of Peer-to-Peer systems to coordinate the two independent motion controllers. The SS2000-PCi-2 includes two controllers in one package, simplifying implementation of such systems. This section also provides a basic overview of larger Host-Slave systems and Peer-to-Peer networks.

The systems engineer using this section must be familiar with the basics of programming the SS2000-PCi-2 and all safety information in Section 2. In addition generally accepted motion programming practices must be used.

In many applications several motors (i.e. "axes") must be coordinated to perform specific tasks. In some applications this coordination must be very tight; all axes must start and stop together, and the path defined by multiple axes must be precisely controlled. In other systems coordination between axes is less critical. The less tightly coordinated systems require coordination of program execution and concurrent motion, but not simultaneous start-stop and path control.

The Host-Slave and Peer-to-Peer systems described in this section are suitable for systems in which path control and precise and concurrent starting and stopping are not required, but concurrent motion and program coordination are desirable.

Host-Slave and Peer-to-Peer systems can be expanded to include many units. In large systems up to 32 controllers can be coordinated through a single serial communication bus, and even more controllers can be integrated into a systems using various control schemes. In the interest of brevity, the details of programming Host-Slave systems and systems involving more than two controls are not detailed in this manual, but the techniques in this section as well as the instructions on Host commands in section 6 form a basis for the design of larger systems.

10.1.1 Applications

Host-Slave and Peer-to-Peer systems consisting of independently programmed controllers are capable of starting each axis in the system within a few milliseconds. Motion of each axis, once initiated, will continue without coordination with the other axis in the system. The stopping time of each axis will depend upon the speed, acceleration, and distance set for each axis. These systems differ from coordinated multi-axis systems in that start times are not as tightly coordinated, each axis completes motion at a different time, and the resultant path of the coordinated axis is not controlled.

Host-Slave and Peer-to-Peer systems are suitable when the motor axes do not need tight coordination. These systems include pick and place applications with point-to-point control, fixture positioning, and machine configuration and setup. Another common family of application is 1½-axis systems in which each axis is moved independently and sequentially under the control of a single program.

Systems that require tight coordination between axes and systems in which the path taken by the coordinated axes must be controlled, must use multi-axis controllers such as the Superior Electric MX2000. Systems that require multi-axis controls include: glue application, sign making, machine tool, flying shear, digital following, and other tightly coordinated multi-axis systems. Point to point systems may also benefit from the increased speed and coordination achieved in a true multi-axis control system.

If after reviewing this manual you have a question about the suitability of the SS2000-PCi-2 for your application, please contact Superior Electric at 860-585-4500.

10.1.2 Overview of Multi-Controller Systems

A simple three axis system is shown in Figure 1. In this system the Primary Control executes a program and co-ordinates that program with two additional controllers.

Coordinated systems require synchronization between the Host or Primary Control and other controllers in the system. In a Host-Slave system the Host initiates actions in each slave. In a Peer-to-Peer system a controller is usually designated as a lead or Primary Control, and the Primary Control initiates action in each subordinate.

Once the commanded action or subroutine is finished the Secondary Control may provide a synchronizing signal to indicate the subroutine or motion is complete.

There are two common methods of communication between controllers in a system:

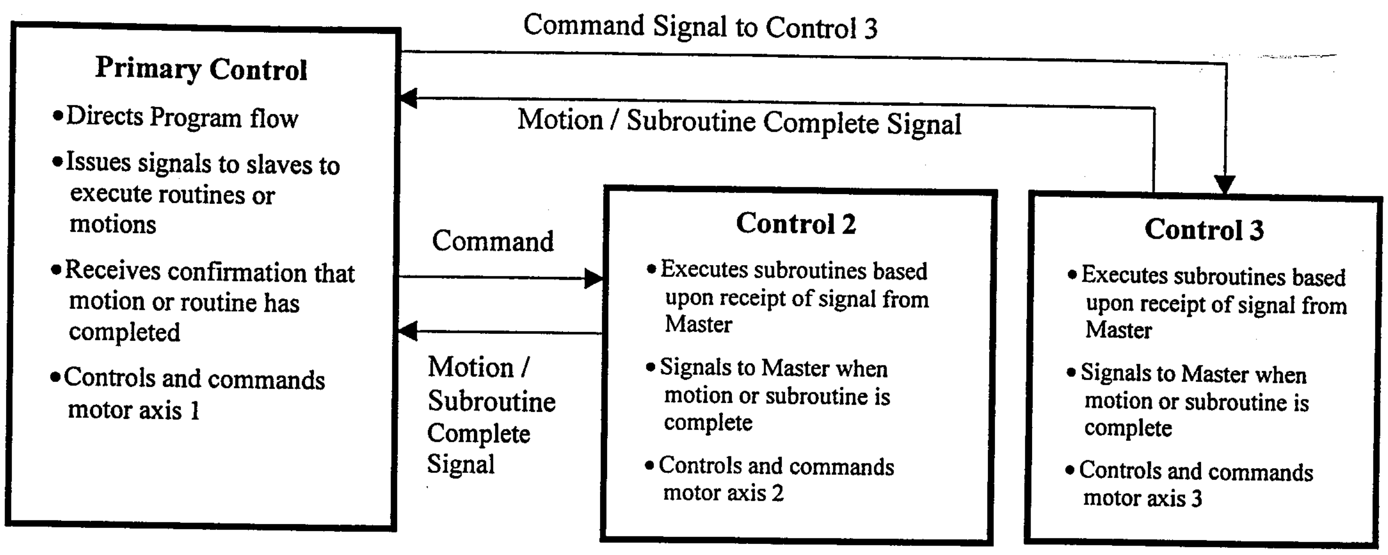
a. I/O communication
b. Serial Bus (RS485) communication

I/O communication:
Each PCi-2 controller has up to 16 inputs and 8 outputs. These inputs and outputs can be used to synchronize the program in a Primary Control with other controls in a Peer-to-Peer system.

Serial Communication
Each PCi-2 controller has two serial ports, the Host Port and the Auxiliary Port. The Host Port (Port #1) is used for programming. The Host port can also initiate motion and control commands directly – see Host commands in Section 6.2 of this manual. In a Host-Slave system the Host controller delivers commands to the slaves through the slave's Host Port.

The Auxiliary Port (Port #2) can be used to communicate between controllers and auxiliary devices. Although the Auxiliary Port cannot respond directly to Host commands such as MOVE and SPEED, the Auxiliary Port can be used to exchange synchronizing signals much as I/O is used for signaling and coordination of controls.

Figure 1 – Typical System


Command Signal to Control 3
Primary Control
•Directs Program flow
•Issues signals to slaves to execute routines or motions
•Receives confirmation that motion or routine has completed
•Controls and commands motor axis 1
Motion / Subroutine Complete Signal
Command
Motion / Subroutine Complete Signal
Control 2
• Executes subroutines based upon receipt of signal from Master
• Signals to Master when motion or subroutine is complete
• Controls and commands motor axis 2
Control 3
• Executes subroutines based upon receipt of signal from Master
• Signals to Master when motion or subroutine is complete
• Controls and commands motor axis 3

In both Host-Slave and Peer-to-Peer systems the Host or Primary Control operates a motor axis in addition to issuing commands to other controls in the system.

10.1.3 Relationship Between Host and Slaves

In a Host-Slave system, all controllers are executing a motion control program. The Slave controls wait until commanded by the host to execute a subroutine that has been previously programmed, or respond directly to a Host command. Each Slave communicates only with the system Host. If the Host is connected to multiple slaves the Host must initiate all communication and query each slave independently for information.

In a Peer-to-Peer system a controller is designated the Primary Control. The Primary Control initiates action in each of the other controls in the system.

10.1.4 Expansion to Multiple Slaves

Using serial communication between the Host and Slave, a single PCi Host control can communicate with and control up to 32 slave controllers.

10.2 Two Controller Systems Using I-O for Synchronization

The balance of this section describes the coordination of two controls in a system. In section 10.2.1 an overview is provided for the development of a system using I/O for synchronization and program coordination. Section 10.3 examines use of the Auxiliary Port and serial communication for program coordination.

In a two-controller system utilizing I/O, at least one input and one output on each controller must be used to synchronize the operation of the two independent controls.

10.2.1 Use of Additional I/O

In some cases additional I/O will be utilized in a system. As an example, in figure 2 the Primary Control (Axis 1 in this example) uses Output #1 to initiate program execution in the Secondary Control (Axis 2 in this example.) Three additional outputs on Axis 1 are used to select for execution one of eight subroutines in the Axis 2 program. See Figure 2. See Section 3.5.2, Input/Output Connections, for I/O connection information.

Figure 2

Example of Mapping I/O points and Secondary Control subroutine:

Output Status on Axis 1 Control (Output #1 Signal Initiates Start of Program in Axis 2 Control)			**Subroutine to Execute in Axis 2 Control**
Output #4	**Output #3**	**Output #2**	
Off	Off	On	Subroutine #1
Off	On	Off	Subroutine #2
Off	On	On	Subroutine #3
On	Off	Off	Subroutine #4
On	Off	On	Subroutine #5
On	On	Off	Subroutine #6
On	On	On	Subroutine #7
Off	Off	Off	Subroutine #8

As systems increase in complexity it may be necessary to change from I/O communication and synchronization to serial bus coordination. Using the serial bus the number of subroutines and combinations is limited only by the program memory available in the control.

10.2.2 Programming Overview

To implement a two independent controller coordinated system using inputs and outputs for control coordination the following steps must be taken.

a) The setup parameters, which include default speeds, programming units, default accelerations and decelerations, maximum speeds, limit switch operation, and other base parameters are established for each control in the system.

b) One controller is selected as the "Primary" Control. The Primary Control operates as the "quarterback" for the system, initiating action in the second control and executing the main system program task.

c) The task required of the second controller is carefully defined and organized.

d) One or more outputs on the Primary Control are selected to signal the execution of required statements or subroutines in the Secondary Control. This output is wired to a selected input on the Secondary Control.

e) Either MCPI or MotionWriter software is used to program each control. The project parameters are set in each controller, and the task program for each control is written. The resulting programs are compiled and downloaded to the controls.

f) Prior to installation the controls are bench tested.

g) The system is wired, all safety items are checked, and the system is started and debugged.

For information about programming and setup of the SS2000PCi-2, please refer to section 3 of this manual. Section 3 includes a startup procedure, connection information, I/O information, and serial bus setup.

Section 5 includes information on the MCPI programming interface, and section 6 includes detailed information on the SEBasic programming language.

10.2.3 Programming Using Inputs and Outputs – Introduction

The diagram and sample program in Figure 3 illustrate a two-controller system that utilizes I/O for coordination. In this example, Axis 1 is selected as the Primary. The Primary Control is typically programmed to accept all operator input and direct the execution of statements or subroutines in the Secondary Control.

10.2.4 Program Flow

In the sample program and system illustrated in Figure 3 Axis 1 is selected as the Primary Control and Axis 2 is selected as the Secondary.

The program begins with definition of variables. The initial statements in programs are typically definitions. Interrupts, if used, would be defined in this section.

Any words or elements following an apostrophe (') represent comments that are not executed as part of the program.

A colon follows program labels (:). Program labels are defined by the programmer, and serve as bookmarks in the sample program – see Sec 5.4.1.1. Labels are also used for subroutines and GOTO statements.

<u>**Axis 1 Program – Initial Motion**</u>

Following definitions, Axis 1 executes a 10-unit move. A **WAITDONE** statement follows. Without **WAITDONE**, the next statement **OUT(1)=1** (turn Output #1 on) would be executed as soon as motion began.

After motion is complete, Output #1 is activated. This signals the Axis 2 control to begin executing statements following label "Motion_Pci_Second:"

<u>**Axis 2 Program**</u>

The Axis 2 control program begins with variable definitions. After the declaration/definition section of the program, further processing of program commands is halted until Input #1 is activated by Output #1 on Axis 1.

The **DO : LOOP UNTIL IN(1)=1** statement halts further program execution until Input #1 is active. The use of DO LOOPS to halt processing of further statements until a condition is met is elaborated on in Section 10.2.6.

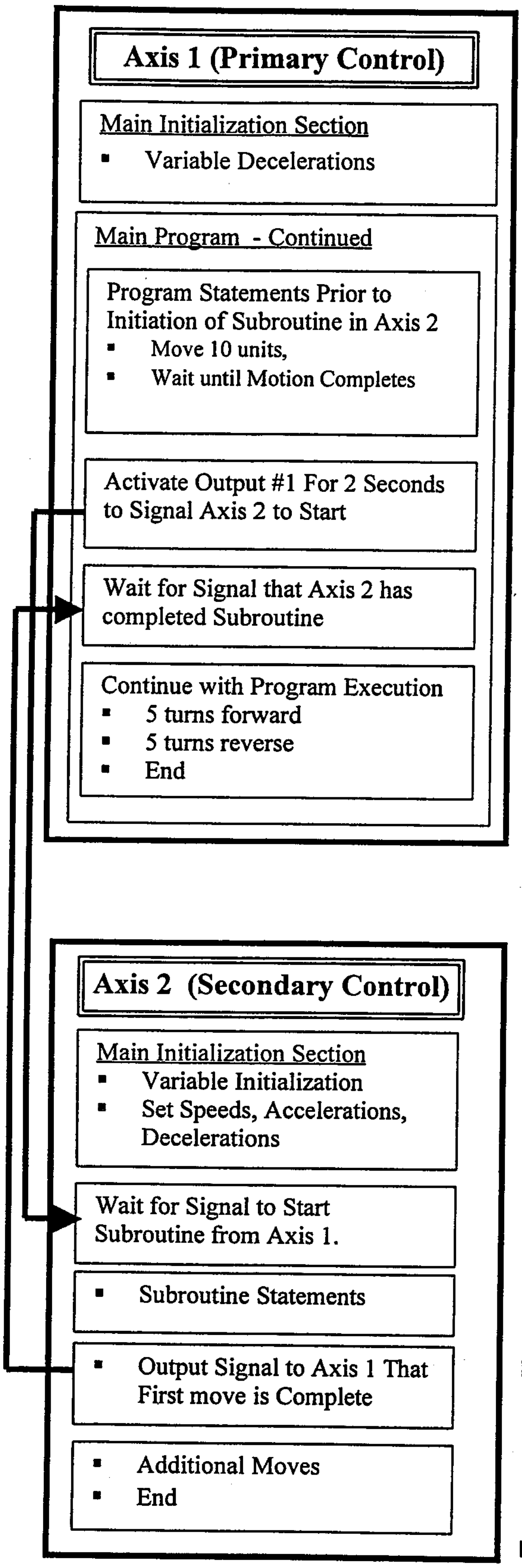
Once input 1 goes active, Axis 2 executes a 10-unit move. A WAITDONE command halts further statement execution until the motion is complete. After motion completes Output #1 on Axis 2 is activated for 2 seconds to signal Axis 1 to proceed.

After Output #1 is activated, Axis 2 executes a Clockwise move of 5 units, followed by a counter-clockwise move of 5 units. This motion occurs while Axis 1 also executes a positive 5-unit move negative 5-unit move.

<u>**Axis 1 Program – Completion of Program**</u>

After setting Output #1 active for 2 seconds thereby signaling Axis 2 to initiate motion, further statement execution in axis 1 is halted until Input #1 receives a signal from Axis 2. The halt in further statement execution is accomplished with a DO LOOP.

After the signal is received, Axis 1 implements a 5 unit move forward and a 5-unit move reverse. The motions in Axis 1 and 2 occur concurrently but are not coordinated.



Axis 1 (Primary Control)
Main Initialization Section
▪ Variable Decelerations
Main Program - Continued
Program Statements Prior to Initiation of Subroutine in Axis 2
▪ Move 10 units,
▪ Wait until Motion Completes
Activate Output #1 For 2 Seconds to Signal Axis 2 to Start
Wait for Signal that Axis 2 has completed Subroutine
Continue with Program Execution
▪ 5 turns forward
▪ 5 turns reverse
▪ End
Axis 2 (Secondary Control)
Main Initialization Section
▪ Variable Initialization
▪ Set Speeds, Accelerations, Decelerations
Wait for Signal to Start Subroutine from Axis 1.
▪ Subroutine Statements
▪ Output Signal to Axis 1 That First move is Complete
▪ Additional Moves
▪ End

Program Code (Axis 1):

```
                                 'Declaring all variables is required

INTEGER a, b, c                  'INTEGER is a whole number, i.e. 1,2,3;
REAL x, y, z                     'A REAL can be a decimal number i.e., 1.543;
STRING A$, B$, C$                'STRING is a character or a word

Motion_PCi_Master:               'Label in program to indicate start of Master (Primary)
      SPEED=5                    'Set Speed to 5 units/sec
      ACCEL=20                   'Set Acceleration to 20 units/sec²
      DECEL=20                   'Set deceleration to 20 units/sec²
      MOVEI=10                   'Commands a move of 10 units for this axis.
      WAITDONE
      OUT(1)=1                   'Activates output #1 of Axis 1 controller
                                 'Output 1 connects to Input 1 of Axis 2
      WAIT=2                     'Time delay of 2 seconds.
      OUT(1)=0                   'Deactivates output #1.

Controller_Pause:
      DO : LOOP UNTIL IN(1)=1    'Holds further program execution until input #1 is active.
                                 'true condition, this program continues to the next line.

Forward:                         'Rotates the motor clockwise.
      MOVEI=5
      WAITDONE

      DO : LOOP UNTIL IN(1)=1    'Holds program cursor until input #1 is activated. At a
                                 'true condition, program continues to the next line.

Reverse:                         'Rotates the motor counter-clockwise.
      MOVEI=-5
      WAITDONE
END
```

Program Code (Axis 2):

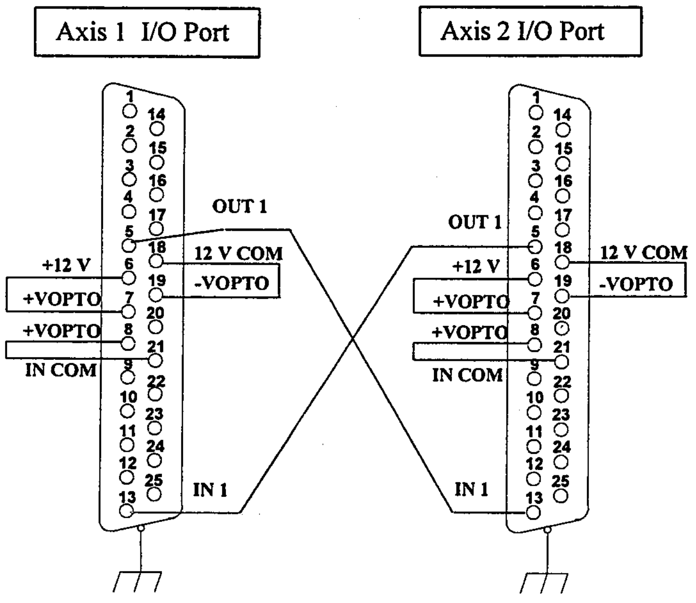
```
INTEGER a, b, c                  'An INTEGER is a whole number, i.e. 1,2,3;
REAL x, y, z                     'A REAL can be a decimal number i.e., 1.543;

Controller_Pause:
      DO : LOOP UNTIL IN(1)=1    'Holds program cursor until input #1 is activated. At a
                                 'true condition, this program continues to next line.
Motion_PCi_Second:
      SPEED=5                    'Set Speed to 5 units/sec
      ACCEL=20                   'Set Acceleration to 20 units/sec²
      DECEL=20                   'Set deceleration to 20 units/sec²
      MOVEI=10                   'Commands a move of 10 units for this axis.
      WAITDONE
      OUT(1)=1                   'Activates output #1 connected to input #1 of the 2nd PCi.
      WAIT=2                     'Time delay of 2 seconds.
      OUT(1)=0                   'Deactivates output #1.

Forward:                         'Rotates the motor clockwise.
      MOVEI=5
      WAITDONE

      DO : LOOP UNTIL IN(1)=1    'Holds program cursor until input #1 is activated.  At a
                                 'true condition, this program continues to next line.

Reverse:                         'Rotates the motor counter-clockwise.
      MOVEI=-5
      WAITDONE
      END
```

10.2.5 Inputs and Outputs – Connection Diagram

In the preceding example, Output #1 on Axis 1 is connected to Input #1 on Axis 2. This connection is shown in figure 4.

Figure 4

Internal Power Supply
Inputs Sinking – Outputs Sinking



It is **STRONGLY** recommended that both ground screws on Axis 1 and Axis 2 connect to the same point.

Please refer to Sec. 3.5.2 - Input/Output connections and Sec. 4.3 - Hardware Equivalent Circuits for additional information.

10.2.6 The Use of DO LOOPS to Trap Execution

Execution of a program in SEBasic does not halt while each statement is executed. This differs from controls programmed with line-by-line interpreters such as the SS2000I controllers.

As a result, the following two program lines initiates motion and while the axis is moving Output #1 is turned on (active).

```
MOVEI = 10        ' Commanded move of 10 units
OUT(1) = 1        'Turn Output #1 on
```

In order to pause execution either a **WAITDONE** or **DO-LOOP** statement block must be used. For example the following three lines of code turn Output #1 on after motion is completed

```
MOVEI = 10        ' Commanded move of 10 units
WAITDONE          'Halt until axis is not BUSY
OUT(1) = 1        'Turn Output #1 on
```

If the programmer wishes to halt execution until motion is complete, a **WAITDONE** statement is best. If other conditions need to be monitored, then a **DO-LOOP** can also be used to halt further statement execution.

The format of a **DO LOOP** statement is

```
DO [condition]
    Statements to Execute
LOOP [condition]
```

The condition can either be at the top (after the **DO**) or at the bottom (After the **LOOP**) but not in both.

The Command

```
DO
LOOP UNTIL IN(1) = 1
```

Loops continuously until input 1 goes active. Statements could be added to the **DO-LOOP** body if certain actions were desired during the loop. In this example, the only requirement is for the program wait or "hang" at this loop until an input is received.

In order to consolidate the command into one line a colon (:), preceded and followed by a space, is used to separate the two lines. Thus the statement

```
DO : LOOP UNTIL IN(1) = 1
```

holds further program execution at that point until input 1 goes high and is identical to the previous command set.

Do-Loops are used in the sample program to halt program execution until the alternate axis controller (either Axis 1 or Axis 2) signals program execution to continue.

For further information about these or other commands refer to Section 6 in this manual.

10.2.7 Input and Output Commands

Both controls use inputs and outputs to provide coordination signals to the alternate axis.

<u>**Inputs:**</u>

The command for reading an input is

```
IN (no.)
```

The number in parenthesis is the input number on the control. Each PCi control has 8 inputs on I/O port on the front of the unit. These inputs are designated as Input #1 through Input #8.

There are also 8 inputs available on the top of the controller. These inputs are on the BCD port. If a BCD is not used, these inputs are available to the program. The BCD inputs are identified as Input #9 through Input #16.

Input #9 through Input #16 are NOT isolated. Care must be taken when using these inputs to insure that no electrical noise enters the system and that the current and voltage rating of these inputs are not exceeded.

In the control setup some inputs can be dedicated to specific purposes, such as limit switch inputs and event triggers.

Inputs 5, 6, and 7 can be configured as RUN, CLEAR, and FEEDHOLD or other user programmable inputs.

Limit switch Input #3 and limit switch Input #4 can ONLY be used as general purpose inputs if the limit switch function is disabled in the program setup. Unless these inputs are disabled as limit switch inputs, activating the inputs will result in a program error.

Any input can be "read" and tested by a program.

When an input is "Active" the value returned by the **IN(x)** command equals 1. When inactive, the value is equal to 0. Thus the statement

```
IN(1) = 1
```

is only "true" when Input #1 is active.

Outputs:

The command for setting an output is

```
OUT(x) = 1
```

The number in parenthesis is the output number. There are four general-purpose outputs on the I/O port on the front of the PCi.

There are also four general-purpose outputs on the BCD port. These outputs are available if the control is not connected to a BCD.

Refer to Section 3.4 and 3.5.2 for additional information on inputs and outputs. Also, refer to Section 5.2.7.6 and Section 5.2.7.7 for additional information on the setup of inputs for dedicated purposes such as limit switch setup and the default setup for RUN/CLEAR/FEEDHOLD inputs.

10.2.8 General Issues

The above example demonstrates some of the techniques employed to synchronize two controllers. The programmer should keep the following in mind.

- Both controls need to be programmed. Either MCPI or MotionWriter must be used for program development. Once a program is developed, it is compiled and downloaded to the applicable control.
- Each control can generate motion simultaneously, but such motions are not coordinated.
- Care must be taken as to the method used to start program execution in each controller. If the controls are set to the defaults – that is input # 5 starts program execution, then input #5 must be activated on each axis for the system to function.
- **If Auto-Start is used, then care must be taken to insure that a motion or program does not inadvertently or unexpectedly start upon power-up.**

10.3 Overview of Serial Bus Synchronized Systems

Just as inputs and outputs are used to coordinate action between two independent controls, the serial ports can also be used to coordinate action. Prior to reading this section, please review section 10.2 - Peer to Peer control using I/O. Many of the same concepts apply to serial coordinated systems as I/O systems and this section does not repeat common information.

10.3.1 The Serial Ports

Each of the controllers in a SS2000-PCi-2 have two serial ports. The "Host" port is used for programming and for Host-Slave operation. See section 6.2. The Auxiliary port can be used for communication with other devices. Either port can be interrogated by the control and used for serial communication with any other serial bus equipped device.

The communication parameters including baud rate and handshake control are configured in software using the MCPI programming interface or MotionWriter. The host port baud rate can be forced to 9600 baud by setting the dip switch on the top of the unit, simplifying communication troubleshooting.

In this section and the examples that follow, it is assumed that the programmer has selected the Auxiliary port for communication between two controllers in a Peer-to-Peer system. This leaves the host port free for programming, debugging operations, host control, and connection to a MMI interface. In this section it is assumed that the programmer has properly set the communication parameters of both SS2000-PCi-2s to enable communication between the two devices.

10.3.2 The Serial Buffer and Input/Output

Each of the serial ports in the SS2000-PCi-2 collects serial bus data and inserts the data received into a serial port buffer. The data in the buffer is read and then removed from the buffer using the INPUT, GETCHAR, and INCHAR commands. To send information from a serial port the PRINT command is used. A brief description of each of these commands follows.

10.3.2.1 The Print Command

The PRINT command sends information from the specified serial port. PRINT#1 prints to the host port, and PRINT#2 prints to the auxiliary port. The syntax of the command is as follows:

PRINT#[port no], Var1 [,Var2, Var3, ...Var3]

port no is either 1 (Host) or 2 (Auxiliary)

Var1 represents a literal value (such as the string "Call Sub 1"), a number, or a defined value.

More than one item can be printed. Use commas to separate each variable, literal, value, or defined system variable (such as SPEED). The following is an example of a valid PRINT command.

PRINT#1, "Speed = ", k

As soon as the PRINT command is executed in the program, the ASCII characters in the print statement are transmitted to the serial bus. In the above example, if integer k = 20, the following ASCII characters are sent to the host port serial bus: Speed = 20

For additional information see Section 6.1.4 – Programming Commands.

10.3.2.2 INPUT# Command

The INPUT# command retrieves characters that have been received into the serial bus buffer. The action of the INPUT# command is dependent upon the presence of a carriage return character (ASCII Character #13) in the buffer when the INPUT# command is executed.

If no Carriage Return Exists in the Buffer

If there is no carriage return character in the buffer, then program execution pauses until a carriage return character reaches the buffer. When a carriage return character reaches the buffer, all of the characters received up to the first carriage return character are read by the INPUT# command. Characters read by the INPUT# command are also removed from the buffer. The inputted string does not include the "carriage return" character. After a carriage return is received into the buffer and the characters are read program execution continues.

If One or More Carriage Returns Exist in the Buffer

If one or more carriage returns exist in the buffer at the time of an input command, only those characters up to the first carriage return are read and removed from the buffer. The remaining characters remain in the buffer until another INPUT# command is executed. After the characters are read program execution continues.

In Figure 5, If the command **INPUT#1, X$** were executed, X$ would contain the string ABC. Input buffer #1 (host buffer) would then appear as in figure 6 below.

If the command **INPUT#1, Y$** followed, Y$ would contain the string SP20.

These two commands could be combined with the single input command

INPUT#1, X$, Y$

In which case X$ and Y$ would contain the strings ABC and SP20 respectively.

Given Figure 5, if the command

INPUT#2, Z$

was executed, then program execution would halt at the INPUT command until a carriage return character was received into the auxiliary port.

Figure 5

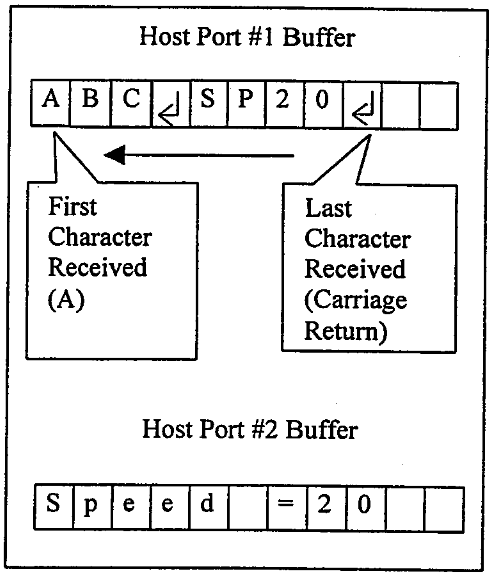
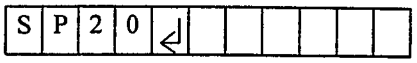


Figure 6

Host Port #1 Buffer after INPUT#1, A$



10.3.3 GETCHAR Command

GETCHAR can also be used to read characters in the input buffer. GETCHAR returns the ASCII numerical value of a single character (first character in) from the serial port buffer.

The format of the GETCHAR command is

Var = GETCHAR(port no)

Var is any integer variable. (port no) is the number of the input port (Host or Auxiliary)

Given Figure 5, if we executed the statement

Z = GETCHAR(1)

Z would equal 65, the decimal equivalent of the ASCII binary code for the letter Capital A.

If more than one character is in the buffer, only one character is read. Once the character is read by the GETCHAR command it is deleted from the buffer.

If there are no characters in the buffer, program execution pauses at the GETCHAR command until a character is received into the serial port buffer.

10.3.4 INCHAR Command

INCHAR is similar to GETCHAR in that INCHAR also returns the ASCII value of characters in the input buffer. INCHAR differs from GETCHAR in that INCHAR does not pause program execution if the buffer is empty. INCHAR returns 0 if there are no characters in the buffer. As a result, INCHAR is often used to flush (remove any characters) from an input buffer.

The syntax of INCHAR is

```
Var = INCHAR(port no)
```

Var is any integer variable, and **port no** is the number of the input port (1- Host, 2 – Auxiliary).

To flush the Auxiliary Port buffer the following statement block could be used.

```
DO
Z = INCHAR(2)
LOOP UNTIL Z = 0
```

For additional information refer to the INCHAR command in Section 6.1.4 – Programming Commands.

10.3.5 CHR$ Command

The CHR$ function returns the character equivalent of an ASCII code. The CHR$ function has two primary uses, decoding the values obtained through the use of the GETCHAR or INCHAR command, and printing or testing for special non-printing characters such as "carriage return" (ASCII 13), "line feed" (ASCII 10), or "bell" (ASCII 7).

In Peer-to-Peer synchronized systems if the INPUT command is used by a control to read control strings transmitted by another control on the system, then a "carriage return" character must terminate the transmitted string for the INPUT command to read it.

For example, to print the string "SUB1" terminated by a "carriage return" character (ASCII code 13), a CHR$(13) must be added to the print statement.

```
PRINT#2, "SUB1";CHR$(13);
```

Use of the Semicolon in Print Statements

In the previous print statement, a semicolon is used as a separator. A comma would insert an additional five "Space" characters between the string "SUB1" and the CHR$(13). A semicolon at the end of the statement suppresses the "linefeed" and "carriage return" characters that are otherwise added to a printed string. This is needed because the additional linefeed would cause the program to advance to the next line of code.

10.3.6 Program Flow

It is recommended that you read Section 10.2.2 - Programming Overview and Section 10.2.4 - Program Flow, prior to reading this section.

The Peer-to-Peer serial bus synchronized system program is similar to the I/O synchronized system program in section 10.2.4. The serial bus synchronized system utilizes serial communication for the synchronization of two independent controls.

In the sample program (figure 7) Axis 1 is the Primary Control and Axis 2 is the Secondary Control. As the Primary Control Axis 1 receives operator input, sends synchronizing signals to Axis 2, and coordinates operation of both controls.

Axis 1 Program – Initial Motion

Following definitions, Axis 1 executes a 10-unit move. A **WAITDONE** statement follows. Without **WAITDONE,** the next statement (PRINT) would execute as motion is initiated.

After motion is complete, the **PRINT#2, "k";CHR$(13);** statement signals the Axis 2 control to begin executing statements following the label "Motion_Pci_Second:" The "k" character insures that the inputted string is not null.

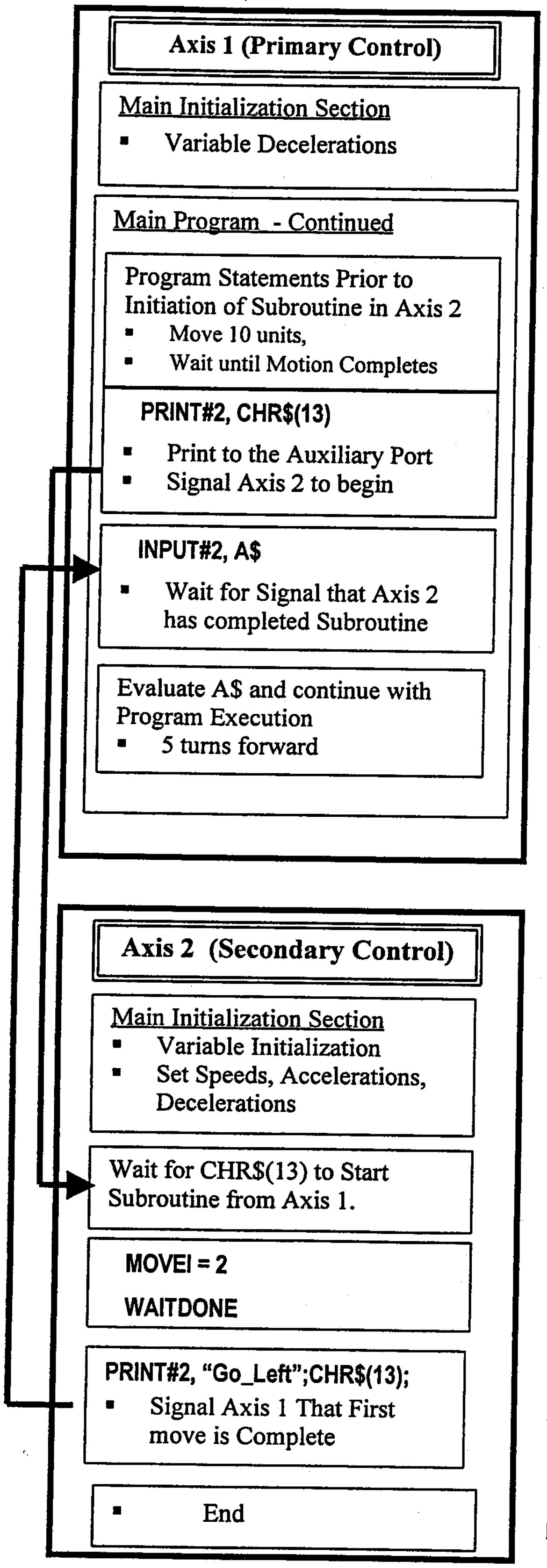
Axis 2 Program

The Axis 2 program begins with variable definitions. After the declaration/definition section of the program, further processing of program commands is halted by the **Input #2, A$** statement until a CHR$(13) is received into the Auxiliary Port buffer.

Once the CHR$(13) character is received in the input buffer, Axis 2 executes a 2 unit move. A **WAITDONE** command halts further statement execution until motion is complete. After motion completes the string "Go_Left" and CHR$(13) is printed to the Auxiliary Port signaling Axis 1 that motion is complete enabling Axis 1 to proceed with program execution.

Axis 1 Program – Completion of Program

After executing the Print command, further program statement execution is halted by **INPUT#2, A$** until a CHR$(13) is received in the Axis 1 Auxiliary Port buffer. Once Axis 2 prints "Go_Left" & CHR$(13) to the serial port, statement execution continues. The IF-THEN statements test A$ and direct program execution to label Forward:, a +5 unit move is executed, and the program terminates.


Axis 1 (Primary Control)
Main Initialization Section
▪ Variable Decelerations
Main Program - Continued
Program Statements Prior to Initiation of Subroutine in Axis 2
▪ Move 10 units,
▪ Wait until Motion Completes
PRINT#2, CHR$(13)
▪ Print to the Auxiliary Port
▪ Signal Axis 2 to begin
INPUT#2, A$
▪ Wait for Signal that Axis 2 has completed Subroutine
Evaluate A$ and continue with Program Execution
▪ 5 turns forward
Axis 2 (Secondary Control)
Main Initialization Section
▪ Variable Initialization
▪ Set Speeds, Accelerations, Decelerations
Wait for CHR$(13) to Start Subroutine from Axis 1.
MOVEI = 2
WAITDONE
PRINT#2, "Go_Left";CHR$(13);
▪ Signal Axis 1 That First move is Complete
▪ End

10.3.6 Sample Program

Program Code (Axis 1):

```
                             ' Declaring all variables is required
INTEGER a, b, c              ' INTEGER is a whole number, i.e. 1,2,3;
REAL x, y, z                 'A REAL can be a decimal number
STRING A$, B$, C$            ' STRING is a character or a word

Motion_PCi_Master:
SPEED=5                      'Set Speed to 5 units/sec
ACCEL=20                     'Set Acceleration to 20 units/sec^2
DECEL=20                     'Set deceleration to 20 units/sec^2
MOVEI=10                     'Commands a move of 10 units this
WAITDONE
PRINT#2,"k";CHR$(13);        'Send a message to the Auxiliary Port to
                             'the other axis. Note: CHR$(13) MUST be used
Controller_Pause:
INPUT#2,A$                   'Holds the program cursor until a word is
                             ' received from the Auxiliary Port
                             'followed by a carriage return.

Data_Evaluated:
    IF A$="Go_Left" THEN GOTO Forward
    IF A$="Go_Right" THEN GOTO Reverse
    IF A$ <> "Go_Left" OR A$ <> "Go_Right" THEN GOTO Controller_Pause

Forward:                     'Rotates the motor clockwise.
    MOVEI=5
    WAITDONE
END

Reverse:                     'Rotates the motor counter-clockwise.
    MOVEI=-5
    WAITDONE
END
```

Program Code (Axis 2):

```
INTEGER a, b, c
REAL x, y, z
STRING A$, B$, C$
SPEED=5                      'Set Speed to 5 units/sec
ACCEL=20                     'Set Acceleration to 20 units/sec^2
DECEL=20                     'Set deceleration to 20 units/sec^2
Controller_Pause:
INPUT#2,A$                   ' Holds the program cursor until a word is
                             ' received from the Auxiliary Port followed
                             ' by a carriage return from the first PCi.
Motion_PCi_Master:
MOVEI=2                      ' Commands a move of 2 units for this axis.
WAITDONE
PRINT#2,"Go_Left";CHR$(13); ' Send a message using the Auxiliary
                             ' Port to the first axis. Note: CHR$(13)
                             ' MUST be used following the semicolon.
END
```

10.4 Host-Slave Systems

The above serial bus coordinated system utilizes the Auxiliary Port for signaling purposes. It is also possible to develop multiple axis systems using the host command capability of the SS2000PCi-2.

In a host-slave system one controller or computer is selected as the Host control. Each of the remaining controls in the system are given a unique ID number from 1 to 32. This number is set using the Device ID switch described in Section 3.2 of this manual.

The Host control can issue "Host Commands" to each of the slaves in the system. Host Commands are listed in Section 6.2.0.

Particular attention must be paid to the bus conflict issues and bus scheduling. In most cases the Host control addresses each controller individually to command a particular move. The Host controller also queries each unit individually to check for position or operation.

10.5 General Issues

A clear understanding of the limitations and capabilities of independently coordinated systems is key to the determination of the suitability of such systems.

The above examples are simple coordinated systems. The approach and strategy employed in these sample systems can readily be expanded to multiple axis systems.

(This page left intentionally blank)