6-1-1994

A Flexible development system for stepper motor based electro-mechanical subassembly design

Joseph Baco

Follow this and additional works at: http://scholarworks.rit.edu/theses

Recommended Citation
Rochester Institute of Technology. Accessed from

This Thesis is brought to you for free and open access by the Thesis/Dissertation Collections at RIT Scholar Works. It has been accepted for inclusion in Theses by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.
A Flexible Development System For Stepper Motor Based Electro-Mechanical Subassembly Design

by

Joseph C. Baco

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of
MASTER OF SCIENCE in
Computer Engineering

Approved By: Dr. Tony Chang - Graduate Advisor

Dr. Roy Czernikowski - Department Chairman

Mr. Glenn Mehalek - Eastman Kodak Sponsor

DEPARTMENT OF COMPUTER ENGINEERING
COLLEGE OF ENGINEERING
ROCHESTER INSTITUTE OF TECHNOLOGY
ROCHESTER, NEW YORK

June 1994

I, Joseph C. Baco, hereby grant permission to the Wallace Memorial Library of Rochester Institute of Technology and to the Clinical Diagnostics Division of Eastman Kodak Company to reproduce my thesis in whole or in part.

Date: June 9, 1994
Abstract

In cooperation with Rochester Institute of Technology and Eastman Kodak Company's Clinical Diagnostics Division, this thesis project describes the design of a PC-based development tool which will aid in the design and testing of electromechanical subassemblies incorporating stepper motors. This tool will integrate the ability to perform digital I/O along with open loop stepper motor control methodologies bundled under a Windows™ based software package to produce a user interface aimed towards the mechanical and electrical engineering development communities. In its maximum configuration, the system consists of 8 single processor control nodes, each of which is capable of executing a control program independently or in cooperation with other nodes. The control programs which run on the nodes are written on the host PC and then downloaded to each of the processor nodes where they are then executed.
Table of Contents

List of Figures ........................................................................................................... 3
Glossary of Terms ......................................................................................................... 5
1. Introduction ................................................................................................................. 7
   1.1 System Architecture .......................................................................................... 8
2. Hardware Design .......................................................................................................... 11
   2.1 The Controller Nodes ....................................................................................... 11
   2.2 The Driver Circuit ............................................................................................. 16
   2.3 Motion Characteristics ....................................................................................... 20
3. Application Software Design ...................................................................................... 22
   3.1 Object Oriented Structure .................................................................................. 22
   3.2 User Interface ...................................................................................................... 26
      3.2.1 The Main Screen ....................................................................................... 27
      3.2.2 Port and Network Control ......................................................................... 30
      3.2.3 Configuring the System ............................................................................. 34
      3.2.4 Manual Control Mode .............................................................................. 36
      3.2.5 Program Control Mode ............................................................................. 41
   3.3 Serial Communications ......................................................................................... 44
      3.3.1 Sending Messages to the Network ............................................................... 45
      3.3.2 Receiving Messages from the Network ....................................................... 47
      3.3.3 Position Updating ..................................................................................... 48
   3.4 Synchronization Mechanisms .............................................................................. 49
      3.4.1 Time Delayed ............................................................................................ 50
      3.4.2 Using the Global Synchronization Line ....................................................... 52
4. System Programming .................................................................................................... 54
   4.1 Programmer's Model ............................................................................................ 54
   4.2 Programming the Nodes ....................................................................................... 57
      4.2.1 Looping ...................................................................................................... 59
      4.2.2 Extended I/O ............................................................................................. 61
      4.2.3 User Command Summary .......................................................................... 62
      4.2.4 System Command Summary ...................................................................... 72
   4.2.5 Sample Programs ............................................................................................ 73
   4.3 Pre-programmed Commands ............................................................................... 75
List of Figures

1.1.1 System architecture overview .................................................. 9
2.1.1 Controller node block diagram .............................................. 12
2.1.2 Network communications block diagram .............................. 13
2.1.3 Typical command message formats .................................. 14
2.1.4 Block diagram of the stepper drive system ....................... 15
2.1.5 Block diagram of the digital I/O system .............................. 16
2.2.1 Block diagram of the driver circuit .................................. 17
2.2.2 Block diagram of the L297 chopper controller .................... 17
2.2.3 Block diagram of the L6203 driver .................................. 18
2.2.4 Excitation patterns in the full step mode ......................... 18
2.2.5 Excitation patterns in the half step mode ......................... 18
2.2.6 Feedback loop of the chopper drive circuit ....................... 19
2.3.1 General motion profile ...................................................... 21
3.1.1 Application software class hierarchy .................................. 23
3.1.2 General support classes .................................................. 26
3.2.1 Main menu screen .......................................................... 27
3.2.2 Communications setup dialog box .................................. 30
3.2.3 Comm. port selection error dialog box ............................. 31
3.2.4 No nodes found dialog box .............................................. 32
3.2.5 Serial comm. error dialog box ....................................... 33
3.2.6 Configuration control menu dialog box .......................... 34
3.2.7 File management dialog box .......................................... 35
3.2.8 The manual control dialog box ....................................... 36
3.2.9 The program control dialog box ..................................... 41
3.3.1 Serial port programming model ...................................... 44
3.3.2 Two level message destination model .............................. 46
3.3.3 PC to the network controllers message model .................. 46
3.3.4 PC to CN message model ................................................ 47
3.3.5 CN to PC message model ................................................ 48
3.4.1 Synchronization delay command location .......................... 51
3.4.2 Global synchronization model ......................................... 52
4.1.1 CN register model .......................................................... 54
4.1.2 Programmer’s memory map .............................................. 57
4.2.1 Extended memory read operations ................................................. 61
4.2.2 Extended memory write operations ................................................. 62
4.2.3 I/O connections for the XY plotter example ...................................... 73
4.2.4 Resulting figure from XY plotter example ......................................... 74
4.3.1 Program file location map .................................................................. 76
Board Layout ......................................................................................... B1
Jumper Settings ..................................................................................... B2
Cable Connections .................................................................................. D1
Glossary of Terms

Controller Node  A controller note is the term used to refer to a CY550 processor board and its associated driver circuit.

CN  See the description of “Controller Node”.

CY233  The CY233 is the part name for the device which handles all of the network communications for a controller node.

CY550  The CY550 is the part name for the microprocessor used in each of the control nodes.

Document  A document (or project) is the name of the file under which the application program stores all of the information specific to a saved session.

Header File  The header file is a file containing the program steps which will be executed on each of the nodes before all others.

High Speed I/O  This is the name given to the digital I/O part which is connected directly to the CY550 processor. It is termed as such since the bits on this port can be directly accessed with a single command.

HS I/O  See the description of “High Speed I/O”.

Motion Profile  The motion profile is the acceleration curve which a stepping motor will follow. It is used to graphically represent the speed of the motor at specific points in time.
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>See the description of “Controller Node”.</td>
</tr>
<tr>
<td>PC</td>
<td>The Personal Computer is an IBM compatible computer system including the monitor, CPU, keyboard, and the mouse.</td>
</tr>
<tr>
<td>Position Wrap Around</td>
<td>Position wrap around is an event that occurs when the 24 bit register which stores the current position of the motor overflows.</td>
</tr>
<tr>
<td>Program File</td>
<td>A program file is a file which contains a list of the program instructions which the user has written.</td>
</tr>
<tr>
<td>Project</td>
<td>See the description of “Document”.</td>
</tr>
<tr>
<td>Quick Buttons</td>
<td>Quick buttons are areas on a window in application program’s main menu which perform some predefined action when they are selected with the left mouse button.</td>
</tr>
<tr>
<td>Screen Buttons</td>
<td>Screen buttons are areas in the application program’s dialog boxes which perform some predefined action when they are selected with the left mouse button.</td>
</tr>
<tr>
<td>Tail File</td>
<td>The tail file is a file which contains program instructions which will be executed immediately after a user program has terminated.</td>
</tr>
<tr>
<td>User File</td>
<td>A user file is file which contains a list of program instructions written by the user.</td>
</tr>
</tbody>
</table>
Chapter 1

1. Introduction

The thesis project presented here differs from the traditional thesis in that much of the theory was developed and applied to produce a fully operational end product. For this reason, the theory used in the project is presented in the areas of the document where it is most applicable.

This project was conceived as part of a joint effort to create a system which would meet the two following objectives:

1. To meet the need in the development community for a development/test bed system which would aid in the design and testing of stepper motor based electro-mechanical subassemblies. Paper feed assemblies, pick and place mechanisms, and positioning systems are examples of such.

2. This system should be designed so as to require a minimum programming background to perform relatively complex operations.

Meeting these design goals would allow both the electrical and mechanical engineering communities to make use of the system with minimum engineering effort.

When setting out to design a system to meet the objectives mentioned above, several issues come into mind. While stepper motors by nature lend themselves to direct microprocessor interface, they also tend to demand fairly accurate timing from the control system in order to achieve smooth operation. As the work load on the controlling processor increases it also becomes increasingly difficult to guarantee consistent response from a basic single processor system. Add in more than one stepper motor, some basic I/O operations, and a reasonably robust user interface and the project becomes a formidable task in both hardware and software design even for today's technology. This creates yet another objective for the development system: Performance of the controller
must degrade "gracefully", or preferably not at all, as the number of motors and I/O lines in the system increases.

With these goals set forth, a discussion of the origins of this project's architecture is provided in the following section.

1.1 System Architecture

After taking into account all of the concerns for smooth stepper operation, expansibility with consistent response, project development time, and the need for a robust user interface, a quick single processor solution to the problem appeared to be out of reach. While it is not absolutely out of the question, a reliable design for a system with such an architecture could require many years of development time. Given these considerations, a multiprocessor solution in which each processor is relatively simple, yet powerful enough to control both a motor and perform some basic I/O tasks seemed to be the best solution. With the widespread use of PC's and workstations today, they seemed to be the two most likely candidates to serve as the foundation of the system. While any of these systems would provide a suitable environment, expansion space, user familiarity, system complexity, portability, and cost left PC as the most logical base from which to design.

Now that a PC host system has been chosen, a method of acquiring the necessary amount of computational power had to be conceived. Adding additional cards onto the PC bus is probably the first and most obvious solution. This method is certainly workable however, one must also take into account the rather large size and power requirements of the driver circuits that must also be included. With this design, heat dissipation would also become an issue. To alleviate some of these technical difficulties, an external driver circuit could be included along with the actual control circuitry which would be housed in the PC. The result is a workable solution to the problem, however several concerns still remain: The number of motors in the system is limited by either the number of slots available on the PC bus or by the number of motors a controller card
could handle. Such an architecture would also require a relatively large number of connections from the PC to the driver boards - on the order of ten per motor, in addition to I/O lines. This design would make the system relatively immobile and prone to cable problems. Additionally, such a system would be relatively fixed in size, and in applications where not all of the motors are being used, it would also be a waste of resources.

In light of these issues and after some extensive research through available products and system designs\(^1\), the system architecture shown in figure 1.1.1 was developed. This MIMD\(^2\) architecture alleviates many of the initial concerns. In this system, the processor in the PC will serve strictly as a graphical user interface engine while coordinating the network at the higher level. It will run a Windows\(^\text{TM}\) application program which adheres to the Windows application user interface standards. This in itself reduces the learning curve needed to use the system by a significant amount due to the fact that a large portion of the engineering community is already familiar with such applications.

Figure 1.1.1: Overview of the final system architecture.
Computational power for the system will be provided by what is termed "Controller Nodes", or CN's for short. Each controller node consists of a single processor, system RAM, thirty or more independent digital I/O lines, and a single stepper driver circuit (see the hardware design section for more details). Communication with the controller nodes is achieved over a standard RS232 connection via a communications port on the PC. In this way, no modifications or internal space restrictions are placed on the host PC. Each node is capable of independently executing a program sequence stored in its local memory.

The Windows application software for the system is written to handle up to 8 controller nodes, however the networking scheme is capable of handling up to 255. Controller nodes may be added or removed from the network without modifications to the hardware or application software, aside from optionally removing the unused nodes from the network. Programming for each of the individual controller nodes is done on the PC. Once the programs are written, they are downloaded to the designated node on the network and executed directly from the individual node’s local memory. While a program is running on a node, very little communication or assistance from the PC is required. In this way, as the number of nodes on the network increases, the only loss in performance occurs in the downloading process as opposed to a loss in stepper performance. Since stepper motors are inherently open loop actuators, some position reporting by the individual nodes was sacrificed in order to support both a scalable system and the synchronization primitives described later.
2. Hardware Design

In the maximum system configuration, there will be eight controller nodes with their associated driver circuitry, a host PC, and a power supply dedicated to supplying power to the controller circuitry and the stepper motors. The nodes will be connected in a ring configuration via an RS232 port on the PC. The following two sections describe the hardware design of the system.

2.1 The Controller Nodes

The simplified block diagram of a controller node (shown with a general stepper driver) can be seen in figure 2.1.1. As shown in the diagram, each node contains all the components of a processing system - a main processor (the CY550), a network interface controller (the CY233), 32K of program memory, and general I/O lines. In addition to these functional blocks, there is a port on the processor which is dedicated to controlling a stepper drive circuit and a set of 4 DAC's intended to adjust the current to the stepper motor windings. On each of the controller nodes, the address decoding circuitry has been provided for six additional chip select lines for future expansion.
The main processor in the system is the CY550 high performance stepper system controller developed by Cybernetic Microsystems. This processor provides all of the necessary computational power to make each node a stand alone entity. Together with the CY233 network interface adapter, also developed by Cybernetic Microsystems, each node is capable of accepting a program of up to 32K in length, and then executing that program without assistance from the PC. This in turn frees up the PC to run the user interface and to simply keep tabs on the progress of each node. Both the CY233 and the CY550 have a complete ASCII command set implemented in their internal programs which allow them to be easily programmed via the RS232 network.

Figure 2.1.2 shows a block diagram of the network communications path from the hardware level of a controller node.
The MAX233 serves as a driver which converts the TTL signals from the network controller into RS232 signals and RS232 signals from the PC or another controller into TTL levels\(^3\). The CY233 network controller and the CY550 have a built-in set of commands which are to be used in a strictly defined format. The CY233 uses the communication parameter jumper settings to select a particular message format and to assign a physical address to the controller node. (See appendix B for address jumper settings). Valid addresses range from $01$ through $FF$. Note that no two nodes should have the same address and that $00$ is an invalid address value. The message format of the CY550 is defined by setting the appropriate bits in the mode register using the "O" command. The format of a typical message can be seen in figure 2.1.3.

![Block diagram of the hardware used to perform network communications.](image)

**Figure 2.1.2:** Block diagram of the hardware used to perform network communications.
Figure 2.1.3: Typical command message formats.

Figure 2.1.4 shows a block diagram of the stepper motor control section of a CN. Since the main function of the controller is to generate control signals to control a stepping motor, a port on the processor is dedicated to performing just that function. The control signals of the port are divided into four major categories indicating the state of motion, the state of limit switches, emergency stop and manual control, and motor direction and speed.

Two of these lines control speed and direction - the “pulse” line and the “direction” lines. Two others called “Stopped” and “Slew” are control signals which indicate the stepping state of the motor. By decoding these two signals, the motor driver board can apply a different current to the motor while it is ramping up to speed, while it is at its maximum speed, and while it is idle. The port also consists of two input signals, “CW Limit” and “CCW Limit” which are intended to provide inputs for limit switches. When the driver circuit drives one of these lines low, the CY550 will ensure that the stepping motor does not turn in that particular direction. If the motor is turning in a particular direction when the respective limit line goes low, the CY550 will simply stop stepping without ramping down. The last two lines on the dedicated port are the “Jog” and the “Inhibit_ABORT” lines. As long as the “Jog” line is left floating, normal stepping
operations will occur. If this line is pulled low, the motor will jog in the clockwise direction and when it is pulled high, the counter-clockwise direction. The rate at which the motor will jog is determined by the value stored in the first rate register (see the F command in section 4.2.1). The “Inhibit_Abort” line may be used as a type of synchronization line or as an emergency stop; that is, if this line is low at the beginning of a motion command, the processor will wait until this line goes high again. If the line goes low while a motion is being performed, the CY550 will ramp down to the first rate value and then cease motion.

Also shown in the figure, are four DAC’s whose outputs also go to the driver circuit. Three of these DAC’s are currently used to set the current to the motor windings, one during an idle state, one during ramp up, and one while the motor its programmed maximum rate. The output voltage of the DAC is determined by the formula:

\[ V = 2.5 \times 255 / \text{DAC value} \]

In addition to the dedicated port and the four DACs, a step mode signal is provided via a signal named USR 7 which is used to select the full step mode when it is low or the half step mode when it is high.

![Block diagram of the hardware used to control the stepper driver.](image)

**Figure 2.1.4:** Block diagram of the hardware used to control the stepper driver.

The last model we will look is that of the digital I/O, shown in figure 2.1.5. Generally speaking, there are two types of digital I/O
available on a controller node, five high speed I/O bits and twenty four extended I/O bits. The high speed I/O bits are provided via a port on the processor and is considered to be high speed I/O since the instruction set allows the bits on this port to be accessed directly. The three byte wide ports of extended I/O bits are provided via a 8255 PIA controller. This I/O is considered to be much slower since these ports must be accessed via the I/O byte register described in section 4.1. If it becomes necessary to add more I/O lines to the system, additional 8255’s can be added through the spare chip select lines already decoded on each of the controller nodes.

![Block diagram of the hardware used to support the digital I/O ports of a node.](image)

**Figure 2.1.5:** Block diagram of the hardware used to support the digital I/O ports of a node.

The resulting memory map of the system is as follows:

- $0000 - $7FFF  Program RAM
- $8000 $DFFF  Unassigned
- $E000  DAC A
- $E001  DAC B
- $E002  DACC
- $E003  DACD
- $F000 $FFFF  PIA

### 2.2 The Driver Circuit

The following section gives a description of the stepper driver circuit which has been provided with each of the controller nodes.
Figure 2.2.1 shows the general block diagram of the stepper driver circuitry. At the heart of this system is SGS L297 stepper motor current controller and sequence generator (figure 2.2.2). This chip is used to form a basic chopping drive circuit which both limits the current to the stepper motor windings and generates the sequences necessary to implement both full and half step modes.

![Block diagram of the stepper drive circuit used in conjunction with the controller nodes.](image)

**Figure 2.2.1:** Block diagram of the stepper drive circuit used in conjunction with the controller nodes.

![Block diagram of the L297 chopper drive controller.](image)

**Figure 2.2.2:** Block diagram of the L297 chopper drive controller.

Four H bridge drivers are used to switch the currents in the stepper motor windings. A block diagram of a single bridge is provided in figure 2.2.3. Together, the L297 and the four L6203 devices provide the majority of the components which are necessary
to implement a chopper drive circuit. A very brief explanation of the operation of a chopper drive circuit is provided here. For a more complete explanation, see the references listed in this paper.

![Block diagram of the L6203 H bridge drivers used to drive the motor windings of the stepper motors](image)

Figure 2.2.3: Block diagram of the L6203 H bridge drivers used to drive the motor windings of the stepper motors.

![Phase sequence of the L297 while in the full step mode](image)

Figure 2.2.4: Phase sequence of the L297 while in the full step mode.

![Phase sequence of the L297 while in the half step mode](image)

Figure 2.2.5: Phase sequence of the L297 while in the half step mode.

The basic feedback path for measuring current in a motor winding is shown in figure 2.2.6. During normal operation, this circuit will attempt to keep the voltage across the drive resistor at a value equal to the voltage at the VRef input by pulsing the enable
lines of the driver circuits. Initially, the circuit will close the path between the motor voltage terminals, allowing energy to flow into the winding. When the voltage across the drive resistor reaches a voltage equal to that of VRef, the flip flop is reset and the drivers are switched in such a way as to short out the winding through the ground plane. At the arrival of the next pulse from the oscillator circuit, the flip flop is set and the drivers are switched back to their original state and energy is again added to the winding and the cycle repeats.

**Figure 2.2.6:** Basic feedback loop of a chopper drive circuit.

As shown in the original block diagram of the driver circuit, an analog multiplexer is used to select one of three voltage levels which will be applied to the VRef pin. Given the DAC value and the value of the drive resistor, the average value for the current in the winding can be determined from the formula:

\[
I = \frac{2.5 \cdot \text{DAC value}}{255} \cdot \frac{1}{\text{Drive Resistor Value}}
\]

Where the drive resistor value is specified in ohms and the resulting current is in amperes.

Given this equation, we can see that the DAC is most useful when the drive resistor value is sized such that the full range of the
DAC is being utilized. This means that as the maximum desired current through a motor winding increases, the optimum value for the drive resistor will decrease. For this reason, the drive resistors on the original circuit boards provided as part of this project have been placed in sockets so that they may easily be changed (see appendix D).

2.3 Motion Characteristics

This section has been included to bring together all of the motion profile issues under one topic. Among the issues covered here are the use of the first rate parameter, the slope parameter, the rate parameter, the first step rate table in appendix A, and a sketch of the acceleration profiles which are generated by the system. Additional information regarding motion profiles and characteristics can be found in the CY550 specification sheets.

The stepper motor motion profile generated by a controller node is defined by three parameters in the processor's internal registers (see section 4.1 for the programmers model), they are the FirstR or first rate parameter, the Rate parameter, and the Slope parameter. First we will take a detailed look at the effect of each of these parameters on the motion profile followed by a look at they are used in conjunction with the optimal acceleration curves built into the CY550.

The FirstR register is used to specify the initial step rate for the motion profile (see figure 2.3.1). The value stored in this register is a pointer into the First Step Rate Table shown in appendix A. This table contains 120 entries, limiting the number of unique initial step rates to those listed there.

The Slope register contains a single byte value which is used to determine the acceleration and deceleration rate of the motion profile. In general, slower accelerations are produced by smaller values of the slope. The formula given below may be used to calculate a rough estimate of the acceleration in steps/second/second.
however, it is only a rough estimate which should be verified using experimental data.

\[
\text{Accel} = \frac{113437}{(256 - \text{Slope})} \quad \text{at 11Mhz clock}
\]

Finally, the Rate parameter is used to define the maximum step rate of the motion profile. The contents of this register represent a time delay which equivalent to the number of processor clock cycles between steps during the maximum step rate phase of the motion profile. This value can either be read from the table in appendix A, or computed as:

\[
\text{Steps/Sec} = \frac{11000000}{(12 \times \text{Rate})}
\]

![Figure 2.3.1: General motion profile characteristics.](image)

As can be seen in figure 2.3.1, the acceleration and deceleration curves are symmetrical and non linear. The general shape of the acceleration curve is as illustrated and will be scaled according to the slope parameter.
3. Application Software Design

This chapter of the text covers all of the issues surrounding the design and use of the application software on the PC. The PC used to design and demonstrate the project is an IBM compatible 486-33 DX2 PC with 32Meg of RAM, 256K of cache RAM, two serial communications ports and a VGA video system. Communications port comm1 was in use for a serial mouse and communications port comm2 was used to communicate with the controller nodes. The application software has been developed using Microsoft Visual C++ version 1.00 and the name of the final version of the application software is "STEPPER Application" version 1.0.

3.1 Object Oriented Structure

Figure 3.1.1 shows the general structure of the application software class hierarchy and figure 3.1.2 shows the general support classes used throughout the software. Boxes which are drawn in grey on these diagrams are classes that are provided by the Visual C++ development software. The other classes that are listed are classes that have been created specifically for this application. As can be seen from the diagram, much of the support for windows and dialog boxes has been provided through the development software. Throughout the course of the application software design, this standard library of classes has helped to cut down on the overall development time. Using some of the object oriented design techniques has also created application software which relatively well structured and straight forward to follow.
CFile - Provides all of the necessary facilities to maintain files and their associated data structures (CFileStatus).

CMenu - Provides all of the necessary facilities to support the pull down menus in the main menu.

CObList - This class provides the functions necessary to maintain an ordered list of objects. Two instances of this class have been used to maintain a list of nodes and to maintain a list of labels.

CPort - This class maintains the communications port, including the message sending and receiving mechanisms.
CNode - Maintains the alias, filename, physical address, and motion status for one node. A list of eight instances of this class make up the model of the system.

CLabelMan & CLabel Together these two classes provide the facilities necessary to maintain a list of label occurrences and label definitions along with their respective memory locations.

CWinApp & CStepperApp - These two classes provide the functionality which is used to maintain the application program within the Windows environment.

CDocTemplate & CSingleDocTemplate - These classes define the template for the stepper project. As implied by the name, the stepper application is a single document (project) application. That is to say that the application software will not allow more than one project to be open at a time since there will be only one network of controllers available.

CDocument & CStepperDoc - These classes provide support for maintaining the information about the current project including file information and some associated data.

CFrameWnd & CMainFrame - These two classes are responsible for drawing and maintaining the main window of the application program.

CControlBar & CStatusBar - These two classes are responsible for drawing and maintaining the status bar at the bottom of the main window.

CView & CStepperView - These two classes provide support for printing and drawing the view of the document. Since this particular application makes use of dialog boxes, these classes are included to support the ability to print program files in the future.

CDialog This class provides all of the functionality necessary to maintain the data for a dialog box. As can be seen from the class hierarchy, several classes are derived from this one.

CFileDialog Provides the dialog box which allows the user to view and select files from any directory on any drive using an interface which is typical of the Windows environment.
CNoNodes - This dialog box is used during the initial setup of the communications port and is utilized when no nodes have been found using the communications parameters.

CSCommError - This dialog box is called whenever a serial communications error has been detected.

CComDlg - This is the dialog box which is used to obtain the communications parameters.

CCommErr - This dialog box is used when the communications parameters have been set inappropriately or the selected port is in use.

CManCDlg - This class maintains all of the data and provides all of the functionality necessary to implement the manual control dialog box.

CPgmDlg - This class maintains all of the data and provides all of the functionality necessary to implement the program control dialog box.

CConfigDlg - This class maintains the data and functionality of the configuration dialog box which allows filename and alias assignments.

CStatic - Instances of this class are used in the dialog boxes to provide static text on the screen.

CButton - Provides dialog boxes with the functionality of screen buttons. Examples of such are the “Done” and “Cancel” buttons.

CLListBox - This class implements the pull down lists used by the dialog boxes. Examples of these are the data rate selection and comm port selection lists.

CComboBox - Instances of this class are used to create areas in the dialog boxes which are capable of displaying and accepting data from the keyboard and program variables. Examples of these are the filename and alias name display areas.
### 3.2 User Interface

This section describes how to use the application software running on the host PC. The name of the Windows executable file is stepper.exe. It is assumed that the user has had some experience using Windows based applications in the past. Areas in the windows shown in the following sections which cause some action to be performed by dragging the mouse pointer on top of them and depressing the left most mouse button are refered to as “Buttons” or “Screen Buttons”.

Once a configuration has been specified, that is, the communications parameters, file names, and alias names have been assigned, the layout may be saved as a file which is refered to as a document or a project. Once saved, the above mentioned data may be retrieved for later use without the need to re-specify all the file
and alias names for the controllers. For more information, see the following sections.

3.2.1 The Main Screen

The main window of the executable is shown in figure 3.2.1. This window consists of both pull down menus (File, Edit, View, Control, and Help) and quick buttons. The button names from left to right are “New Document”, “Open Document”, “Save Document”, “Port Configuration”, “File Configuration”, “Manual Control”, and “Program Control”. The functionality of each of the quick buttons is also available via the Control pull-down menu. At the bottom left of the screen is a long rectangular box in which a brief explanation of the functionality of the selected button will appear. This is a quick way of determining the functionality of each of the quick buttons. Following the figure is a brief explanation of the functionality of each of the quick buttons.

![Figure 3.2.1: The main window of the PC application “stepper.exe”](image)

The “New Document” button will first prompt the user to save changes to the current document if any have been made since
the last save. Once the original document is closed, a new blank document will be opened.

The “Open Document” button will first prompt the user to save changes to the current document if any have been made since the last save. Once the original document is saved, a file management window will open which will allow a previously saved document to be opened as the current document.

The “Save Document” button will save any changes to the currently opened document under the document name given previously. If a name has not been previously given, a dialog box will prompt the user for a new file name.

The “Port Configuration” button will open up a dialog box which will allow the user to change the port and data rates. See section 3.2.2 for more details.

The “File Configuration” button will open up the file configuration dialog box which will allow the user to change the alias names and associated program files. See section 3.2.3 for more details.

The “Manual Control” button will open up the manual control window which will allow the user to control the motors on the network via controls on the screen. See section 3.2.4 for more details.

The “Program Control” button will open up a window which will allow the user to run program files on each of the nodes. See section 3.2.5 for more details.

The “Print” quick button will currently print a blank document. This quick button has been put in place in the event that an editor is added in the future.
The "Help" button will open up a dialog box which will display the name of the application program, the author, and the software version number.

The "File" menu contains commands which are specific to the document which is in use. The "New" command is synonymous with the "New Document" quick button, the "Open" command is synonymous with the "Open Document" quick button, and the "Save" and "Save As" commands are synonymous with the "Save Document" quick button. The "Print", "Print Preview", and "Print Setup..." commands have not been implemented at this time. Below the printing commands, the application will list the last 4 documents which have been used. For convenience, the user may select any one of these to be the current document rather than using the "Open" command. At the bottom of the "File" menu is the exit command which is used to exit from the application program.

The "Edit" menu to date is not used by the application program. It has been added on in the event that the ability to edit user programs directly is added on in the future.

The "View" menu allows the user to configure the main menu window. The "Toolbar" command toggles the toolbar status, making the quick buttons visible or invisible. The "Status Bar" command will alternatively display or hide the status bar which displays brief help text at the bottom of the screen. This status bar also indicates the status of the caps lock, the num lock and the scroll lock buttons on the keyboard.

The "Control" menu provides access to the four main functions of the application program. The "Comm Port" command is synonymous with the "Port Configuration" quick button, the "Configure" command is synonymous with the "File Configuration" quick button, the "Manual" command is synonymous with the "Manual Control" quick button, and the "Program" command is synonymous with the "Program Control" quick button.

The "Help" menu has a single command "About Stepper ..."] which opens up a dialog box displaying information about the
application program itself, including the application name, the author’s name, and the version number of the software.

3.2.2 Port and Network Control

Before the system can be used, the PC communications port to which the controllers are connected must be specified, along with the data rate at which the network will be run. This is done in the communications setup dialog box as selected by pressing the port configuration button on the main menu (also available in the Configuration pull-down menu). Since this configuration must be done before any communications can begin, this dialog box will appear automatically before the file configuration, manual control, or program control modes can be entered. Once it is done, there will be no need to do it again unless the number of nodes on the network has changed or the data rate has been altered.

The communications dialog box can be seen in figure 3.2.2. When the box first appears, the status indicator will read “Awaiting port parameters”. To perform the configuration, first select the communications port to which the nodes are connected by clicking the mouse anywhere in the comm port indicator. A list of two selections will be presented. Select the appropriate comm port number (1 or 2). The data rate selector operates in a similar manner. Select either 19200 or 9600. The jumpers on the boards have been initially set to 19200 baud, and must be the same for all nodes on the network. (To change the data rate setting on a node, see appendix B).

![Communications Setup](image)

Figure 3.2.2: The communications setup dialog box.
Once the comm port and data rate have been selected, clicking on the “Find Nodes” button will cause the application software to search the selected comm port for all nodes which are running at the specified data rate. The search is accomplished though the use of the network “S” command as described in section 4.2.1. While the search is being done, the status will read “Searching for nodes ....”. If the communications parameters are set correctly and power has been applied to the CN’s, the application program will determine the physical address of each node automatically. At this point, the done button should be pressed and the network will be ready for use.

During this process, there are several events which may prevent the network from being initialized properly. A set of three dialog boxes have been created which serve to describe the error. The first and perhaps the most common error during the configuration process is one in which the communications parameters have been specified incorrectly. The dialog box which indicates this error is shown in figure 3.2.3. Several errors may cause this dialog box to be presented. The first is an incorrect data rate or the selected communications port does not exist. The second occurs when the selected port exists however it is in use by another device, typically the mouse. When this error occurs, click on the OK button and correct the communications parameters.

![Communications Error!](image)

**Figure 3.2.3:** Dialog box which indicates the communications parameters have been incorrectly specified.

Another type of error which may occur is one in which the selected communications port exists and is not in use by another device however no CN’s have responded to the search. This error may be caused when the CN’s are connected to a port other than the one specified, there is a break in the network connections, or when
power to the nodes has not been turned on. When this dialog box appears, click on the "OK" button and check all of the points described above. If all of the parameters, the network, and the power levels seem to be correct, push the reset button on each of the nodes and click on the "Find Nodes" button once again.

<table>
<thead>
<tr>
<th>Error: No nodes found.</th>
</tr>
</thead>
<tbody>
<tr>
<td>No nodes have been found on the selected communications port. Please check the power to the nodes and/or the communications port settings.</td>
</tr>
</tbody>
</table>

Figure 3.2.4: Dialog box which indicates that the port exists however no CN’s have responded.

The error dialog box shown in figure 3.2.5 is a communications error which may occur at any time during the operation of the system. Several conditions may cause this error to occur. While in manual control mode, this error may occur when the node under control stops responding to position update messages or, in cases where an illegal motion is attempted, position update message may be simply echoed back. In either of these cases, it is best to first press the reset button on the node which was being operated and see if communication is restored. If the error occurs immediately after pressing the "OK" button, try resetting all of the nodes on the network and pressing the "OK" button again. Of course, we’re assuming here that all of the network connections are in place and that each CN has 5 volts applied to it.
Serial Communications Error.

Error: The PC has lost communication with the network.

1. Check all network connections.
2. Check the power to each node.
3. Press the reset button on each of the nodes and hit the "OK" button below.

Figure 3.2.5: Dialog box indicating that a serial communications error as described in the text has occurred.

While in the program control mode, this error may occur for the same reasons as in manual control mode in addition to a few others. While a program is running on a node, there exists several instructions which may interfere with the position update mechanism (see the command summary in section 4.2.1). If one of these instructions is being executed when a position update is requested, there is a good chance that the query will be ignored or echoed back to the PC. In either case, it impossible for the PC to tell whether communication has been lost or one of these instructions is being executed and hence it must be interpreted as an error. Also, if more than three consecutive position queries go unanswered, the PC will assume that there is a break in the network connection and thus generate this error.

To clear this error, check all of the network connections to the nodes, as well as the power source. Press the reset button on each node and reload the files. Note that once this error has occurred, all of the nodes on the network will be re-initialized and the respective program files on each node must be restarted. For the reasons mentioned previously and for synchronization reasons, it is highly recommended that the position query mechanism be left turned off unless it is needed.
3.2.3 Configuring the System

If the system is going to be run under program control, the configuration menu can be used to assign program files and aliases to each of the CN’s on the network. The dialog box, shown in figure 3.2.6, can be opened by pressing the “File Configuration” button or via the “Control” pull-down menu in the main window.

The “Physical Address” column of the dialog box indicates the physical address in hexadecimal notation, of each of the nodes in the order they are physically connected to the network. These addresses are determined as result of the communications setup procedure. These addresses are set in the jumpers of the CN’s and may not be changed from the PC (see appendix B). The next column, the “Alias” column, is a column in which the user may choose to assign an alias name to each of the nodes. This alias name is useful since a meaningful title such as “Film Feed Motor” may be assigned to particular CN. Once an alias is given, it is used to reference that CN throughout the program. The default alias for each node is the hexadecimal representation of the physical address of that node.

<table>
<thead>
<tr>
<th>Physical Address</th>
<th>Alias</th>
<th>Program Filename</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>

![Figure 3.2.6: The configuration control menu which is used to assign program files and alias names.](image)

The “Program Filename” column indicates the full pathname to the program file which will be run on the corresponding node.
This pathname may be changed only by pressing the adjacent button under the "Change" column. Pressing a Change button will cause the dialog box shown in figure 3.2.7 to be displayed. This dialog box is particularly useful in finding and selecting a program file for use on a node. Selection is done by first selecting the disk drive in the lower right hand corner, which contains the program files which will be used. The default drive is the C drive. Once this is done, the path through the subdirectories may be setup by opening and closing the file folders as appropriate under the "directories" header located at approximately the upper center of the dialog box. Under the "Filename" heading at the upper left hand corner of the dialog box, the file selected from the list below will appear. Use the mouse pointer to select the desired file from the list and press the "OK" button. Optionally, the "List of Files Type" field may be set to filter out certain types of files in directories which contain large numbers of files. If a "Change" button has been accidentally pressed, the "Cancel" button will exit from the dialog box without modifying the "Filename" field in the configuration.

**Figure 3.2.7:** The file management dialog box which will allow the user to select a program file from any available drive.
3.2.4 Manual Control Mode

The manual control screen shown in figure 3.2.8 is opened by pressing the manual control button or by selecting “Manual” from the “Control” menu in the main window. The manual control mode allows each of the CN’s on the network to be operated without the use of program files. It is intended for use in applications where simple motions will be performed and full functionality of the system, such as digital I/O, will not be needed.

The controls in this window provide the means to control a motor connected to a CN in two conceptually distinct modes - continuous, and absolute. This mode is always displayed in the small box under the current node’s alias name. The following paragraphs describe the operation of each of the controls shown in the window. Note that the functionality of some of the controls may change as a result of the current mode of motion.

![Manual Control Menu](image)

**Figure 3.2.8: The manual control dialog box.**

The node name control displays the alias name of the node to which all of the commands initiated from
this window will be sent. The small arrow buttons to the right of the control can be used to change from one node to the next. The small box under the node name will always display the current motion status of the selected node - “Motion: Inactive”, “Motion: Continuous”, or “Motion: Absolute”. Note that if the current node is changed while motion is occurring, that motion will continue until the desired position is reached or the node is explicitly halted.

The drive board resistor indicator on the screen has the same operation no matter what the motion state of the motor. This value should be changed to reflect the value of the feedback resistors installed on the driver circuit boards. Since it is used for current calculations and boundary condition calculations, it is important that this value be kept accurate. It is entered by simply selecting the text contained within it and typing in a new value via the keyboard.

The DAC value controls perform the same operation no matter what the state of the motor. Each of the three DAC values may be changed independently via these controls. Pressing a “<“ button will retrieve the respective DAC value, decrement it by one if it is greater than 0, update the value of the current on the screen, and issue the commands necessary to update the corresponding DAC value on the selected node. The “>“ buttons will perform the same function however they will increment the DAC value by one, provided doing so will not cause the calculated current to rise above 3.5 amps. A DAC value may also be entered directly by simply selecting the corresponding value under the “DAC Value” column and typing in a new value. If the typed value exceeds a corresponding current of 3.5 amps, the value will be set to the maximum DAC value which does not exceed this current. Default DAC values are all 0.
The Exit button will cause the application program to exit the manual control mode. Before doing so, all motion on the nodes will be halted.

The Stop Node button will cause any type of motion in the associated stepper to cease. All other nodes are unaffected.

The Stop All Motion will cause both types of motion on all nodes on the network to cease.

The Step Mode control is used to issue the appropriate commands to set user bit #7 to the proper state for selecting full and half step modes. The display will always indicate the current step mode. The mode may be changed only by pressing the Half of Full buttons. This control performs the same function for both types of motion.

The step direction control indicates the current step direction while stepping in the continuous mode or after the execution of a single step command. If the either direction button is pressed while a continuous motion is being performed, the controller will ramp down the motor using the current motion profile. Once the ramp down is completed, the direction line will be updated to the new state. If motion is being performed in the absolute mode, this control will have no effect on the motion. Also note that while performing absolute motion, the indicated direction may not be valid.

The single step buttons may be pressed at any time. If the motor is currently in absolute or continuous stepping modes, the controller will ramp down using the current motion profile and perform the single step. After the single step is performed, the motion state will be idle. The step direction control will be updated to reflect the direction of the single step.
Slope: The slope parameter determines the acceleration profile of any motion. Values may be entered by selecting the current value and typing in a new one via the keyboard. In general, the lower this value is, the slower the acceleration. The actual acceleration profile of the motor is generally non-linear and should therefore be determined experimentally (See section 2.3 for details). During absolute motions, this value can be entered via the keyboard only and then updated using the “Go To” button on the screen. During continuous motions, the value is updated by pressing the “Go To” button or through the use of the acceleration control group.

First Step Rate: The first step rate value can be updated via this control during absolute motions. The value on the controller is updated using the “Go To” button on the screen. Its value is a pointer into the First Step Rate Table in appendix A. For more information on how this parameter affects the motion profile, see section 2.3. During continuous motions, changing this value will have no effect.

Slew Rate: The slew rate field represents the maximum step rate for a motion profile (See section 2.3). During absolute motions, this value can be updated via the “Go To” button. During continuous motions, this value may be changed via the “Step Rate” controls group or typed in. The current value of the slew rate is converted to steps per second and displayed under the Step Rate group of controls.

Current Position: This is a read only control which is updated automatically by the application with the contents of the current node’s position register.

Desired Position: The desired position control is used to specify the new absolute position when performing an absolute motion. When the value has been entered, the “Go To” button will initiate the motion.

During absolute motions, the reset button will cause the motor to ramp down using the current motion profile (See section 2.3). The contents of the position register of the node will then be
set to zero. Pressing the reset button during continuous motions will have no effect.

During absolute motions, this button will cause all motion to cease. It will then download all the DAC values and the motion profile parameters. Finally a command sequence is sent which will cause the motor to use the motion profile to get to the desired position. During continuous motions, this button will have no effect. While in the absolute stepping mode, the DAC values may be updated without affecting the motion.

The Step Rate control group is a group of buttons which will control the maximum step rate during continuous motions. The smaller buttons labeled with "<" and ">" will increment or decrement the current maximum step rate by 1 while updating the value in the controller's register. The buttons labeled "<<" and ">>" will increment or decrement the current maximum step rate by increments of ten. The small box below these controls always indicates the current maximum step rate in steps per second. During absolute motions, these controls will have no effect.

The Acceleration control group is a group of buttons which will control the acceleration value during continuous motions. The smaller buttons labeled "<" and ">" will increment or decrement the current acceleration value in increments of one. The larger buttons labeled "<<" and ">>" will perform the same functionality in increments of ten. The small box below the buttons always indicates the current acceleration parameter. During absolute motions, these controls will have no effect.

The continuous button is used to initiate a continuous motion. When this button is pressed, all of the motion
profile parameters are set, along with the current DAC values. The current direction displayed by the direction controls is then used to determine the direction of the continuous motion. Motion is initiated using the current motion profile. During continuous motions, the maximum step rate, acceleration parameter, and DAC values may be changed without stopping the motor.

### 3.2.5 Program Control Mode

The program control mode dialog box is opened by clicking on the “Program Control” button or by selecting “Program” from the “Control” menu in the main window. The program control dialog box is shown in figure 3.2.9.

![Program Control Menu](image)

**Figure 3.2.9:** The program control dialog box.

The controls in the dialog box are arranged in a manner similar to that of the file configuration box. Controls with the same functionality are arranged in vertical columns while controls for a particular node are arranged horizontally. This provides for a logical grouping of functionality. Before describing each of the columns of controls, it is more appropriate to describe the functionality of the buttons positioned along the bottom of the menu.
Pressing the “Load Files” button will cause the application program to download the header and tail files to each node, along with the individual program file designated for each node. Nodes which do not have a specified file will be loaded with a header and tail file along with a null program file. The status indicator bar will indicate whether or not the program files have been loaded successfully and if not, it will give an indication as to the file and line number on which the error occurred. For more information on the status bar as files are being loaded, see section 4.2.

The “Position Update” button will toggle the position update mechanism on and off. The current state of this mechanism is indicated in the box next to the button. Note that some program instructions may interfere with the position updating mechanism (See section 4.2.1) and it should therefore only be used when these instructions are not present in any of the user programs. Also note that the position updating mechanism is likely to interfere with the accuracy of global synchronization and should not be used in cases where this is an issue.

The “Done” button will cause a stop command to be issued to all of the controllers, followed by returning the application to the main window. If alias names have been modified, the alias names become permanent.

The “Cancel” button will also dispatch stop commands and return the application to the main menu however any changes to the alias names will not be saved.

The column labeled “Driver Name” displays the alias name assigned to each of the controllers. Note that the alias names may be changed here without going back into the Configuration Control menu. If the “Cancel” button is used to exit from this dialog box, changes to the alias names will be lost.

The column labeled “Filename” displays the full pathname of the program file which has been assigned to this node in the Configuration Control menu box. These filenames are read only from this window and may be changed only while in the Configuration Control box.
The next column of buttons is the “Run” and “Run All” buttons. Pressing a “Run” button will cause the associated node to begin program execution from where it has left off. In cases where the files have just been loaded or where the corresponding reset button had been pressed, program execution will begin at the beginning of the header file. In cases where the stop button had been pressed, program execution will continue from where it previously left off. Note that not all commands may be restarted in this manner and it is suggested that program execution be initiated only after a “Reset” or “Reset All” button has been pressed. Pressing the “Run All” button will cause a start command to be issued to all controllers on the network. See section 4.1 for complete details.

The column of “Stop” and “Stop All” buttons is used to halt program execution on the respective node. Pressing this button simply stops program execution and does not reset the program counter on each of the nodes.

The column of “Reset” and “Reset All” buttons will perform the same function as the stop buttons however they will also reset the value of the program counter. If a program is not already running, only the value of the program counter is reset.

The next two columns of controls are used by the position updating mechanism to display the current position of the stepping motor and its current maximum step rate. Due to the inherent delay of the RS232 network, these values rarely represent the precise position at the moment they are displayed, especially in cases where high stepping rates are used.

Note that because the CY550 processor must finish processing its current command before any others can be processed, commands that cause a “block”, such as those that wait for a user input bit to change state, will not allow position update messages to be processed nor will they allow stop, reset, or start commands to be processed until the condition of the instruction has been satisfied. During this time, any such commands issued from the PC will be ignored completely and will have to be re-issued by the user. In some cases this may cause the PC to call a communication error since the
message does not bounce and no reply is generated, we must assume that the network has been broken. If the processor is "stuck" on this type of instruction, the reset button for that node must be pressed in order to interrupt the program execution and regain control of the node. Commands which are listed in section 4.2.1 as commands which will interfere with the position update mechanism will also interfere with the start, stop, and reset functions.

3.3 Serial Communications

Communication with the network is accomplished via one of two serial communications ports on the PC (COM1 or COM2). Since the PC's program is a Windows application, we may use the Windows serial port drivers and system calls to create a serial communications package. A brief model of the serial communications software model is shown in figure 3.3.1.

![Diagram of serial port programming model](image)

**Figure 3.3.1:** Basic serial port programming model.

Upon opening the serial port, a transmit and a receive buffer are created along with a port error status data structure and a queue status data structure. Both of the buffers are interfaced to the communications port hardware via the serial port interrupts on the PC. The status of the port and the status of the buffers are available through a status data structure. Within this structure are the number of characters in each queue and the last serial communications error to occur. (As of this writing, communications occurs at a data rate of 19200 with no parity, 8 data bits and 1 stop bit).
Once the port has been opened and the appropriate communications parameters have been set, messages may sent and received from the port through the WriteComm and ReadComm procedure calls. The entire message processing mechanism on the PC is based on these two routines as described in the following two sections. These two sections describe the conceptual process by which messages are handled and assume that the initial messages to be sent are generated somewhere within the application program and messages that are received from the network are the result of some previous message generated from within the application program.

3.3.1 Sending Messages to the Network

Messages being sent to the network from the PC have two basic formats, that is, messages that are destined for the CY233 network controllers and messages which are destined for a particular controller. All messages in the system are destined for one of these two conceptually different targets as depicted in figure 3.3.2. Messages sent to the controllers generally go to every controller on the network and return to the PC where as messages sent to a particular node are only sent to that node and stop there.

In addition to these two basic formats, messages are further broken into two categories - those which expect to generate a reply message on the network (ie. position queries) and those which do not (ie. a stop motion command). First, we'll look at the general message sending mechanism followed by a brief explanation of how messages expecting a reply work in conjunction with receiving messages from the network.
Figure 3.3.2: Two level message destination model.

Figure 3.3.3 depicts the process by which a message destined for the CY233 network controllers is handled. All that needs to be added to these messages before placing them in the FIFO output queue is the "cr" message terminator. This message will then be placed in the outgoing message queue. Generally speaking, messages destined for the controller nodes will travel to every node on the network and return to the PC and thus they inherently generate a reply message which is simply the original message.

Figure 3.3.3: Sending a message destined for the network controllers.

Messages which are destined for a particular controller node are handled in a slightly different manner as is shown in figure 3.3.4. In addition to the message terminator "cr", a header containing the address of the destination node and a message direction character of "W" is added to the front of the message. The address of the destination node is used of course to determine which node on the...
network will receive this message. The "W" command tells the network controllers that this message is to be "written" to the node with a matching address. When the network controller with the matching address receives the message, it will pass the original command and the message terminator to its associated node controller. In effect, this strips off the message header before it reaches the node controller and thus a valid CY550 command is generated.

![Diagram](https://via.placeholder.com/150)

**Figure 3.3.4:** Sending a message destined for a particular processing node.

### 3.3.2 Receiving Messages from the Network

The mechanism used to receive messages is directly dependent on whether or not a message which was previously sent expects to receive a reply. Figure 3.3.5 depicts the general message receiving algorithm. Since the input buffer is a character based queue rather than message based, an additional layer of software has been added between the input queue and the message processing software. This additional layer takes characters from the input queue and groups them into complete messages which can then be processed by the system. Only complete messages are generated by this character grouping and thus partial messages are not available until they are received entirely.

When a message which expects a reply is sent out to the network, whether it be a network controller message or a node message, a record is made indicating that we should expect to see a return message from the port. A timer is then set to keep track of
the amount of time a message has been out. If after a predetermined amount of time (determined by the mode in which the network is placed) a reply message has not been returned, the message is considered to be lost. Since all messages in the system should be accounted for, in most cases this event constitutes a serial communications error which is described in more detail in section 3.2.2. If no messages are expected, the characters which are received are grouped into messages and monitored for serial communications errors which are also described in greater detail in section 3.2.2.

![Diagram](image)

**Figure 3.3.5:** Receiving a reply message from a node controller.

### 3.3.3 Position Updating

The position update mechanism for both the manual and program control nodes is unique in that the message serves two purposes. Its first purpose is to retrieve the current position of the stepper motor from a specific node. Its second purpose is to check the status of network communications. Since under some circumstances (mostly when running a node processor at high step rates) it is possible for a node to ignore a position query message, the software must be tolerant of this. In most cases, it is impossible to tell whether the network connection has been broken or if a node is simply ignoring the query.
As described in the previous section, a lost reply message usually constitutes a serial communications error; however in this case that is not always acceptable. To compensate for this, the PC keeps track of the number of position queries that have not been answered since the last complete update had been received. If more than 3 contiguous queries have gone unanswered over a period of approximately 2.5 seconds in program mode or 1.8 seconds in manual mode, we assume that one of two events has occurred: either the PC has lost communication with the node being queried or the network connections have been broken. It is only after these conditions are satisfied that an error message is generated. Notice however, that this does not completely eliminate false serial error communications error messages from being generated. In cases where the user's program has executed a command which disables the position query mechanism for a considerable amount of time, false communications errors will in fact be generated. It comes down to a trade-off between how often the position display is updated and how many consecutive lost messages constitute a serial communications error. See section 3.2.2 for more details on this subject.

3.4 Synchronization Mechanisms

Given the system architecture and inherent delay of the RS232 network, it is impossible for the PC to initiate program execution on all of the nodes simultaneously. As the number of nodes on the network increases, this problem will become more obvious. For example, if we simply sent the "X" command (begin program execution from local memory) to all of the nodes, the delay between the time the last node on the network began execution and the time the first node begins execution (assuming we initiate execution on the last node first, giving us the best case using this method) would be:
PC

Nodes

\( t = 0 \) Send "R08E-" \( t = 0.00156 \) Node 1 Trans. "R08" to node 2.

\( t = 0.0026 \) Send "R07E-" \( t = 0.00312 \) Node 2 Trans. "R08".

\( t = 0.0052 \) Send "R06E-" \( t = 0.00468 \) Node 3 Trans. "R08".

\( t = 0.0078 \) Send "R05E-" \( t = 0.00624 \) Node 4 Trans. "R08".

\( t = 0.0104 \) Send "R04E-" \( t = 0.0078 \) Node 5 Trans. "R08".

\( t = 0.013 \) Send "R03E-" \( t = 0.00936 \) Node 6 Trans. "R08".

\( t = 0.0156 \) Send "R02E-" \( t = 0.01092 \) Node 7 Trans. "R08".

\( t = 0.0182 \) Send "R01E-" \( t = 0.01248 \) Node 8 Starts.

\( t = 0.0208 \) Finished Node 1 starts.

Where \( t \) is in seconds.

Worst case discrepancy for 8 nodes: .00832 seconds between the time node 8 begins execution and the time node 1 begins.

Although this delay is acceptable for most applications, one would not be hard pressed to conceive of a situation where such a large delay is unacceptable. In light of this problem, two methods for reducing the amount of delay between nodes are introduced and described in the following two sections. One method, called the time delayed method, produces better results than the direct method however the nodes are still not perfectly in sync. The second method, called the global synchronization method, produces the best synchronization we can achieve with this architecture.

3.4.1 Time Delayed

The timed delayed method makes use of the delay command ("D") of the user's command set (see the "D" command in System Programming for more details). This is the method used by the application program to synchronize the nodes. The theory of this method is as follows: given the speed of the network and the time delay through each of the network controllers, a time delay value
which will cause all of the nodes to start at approximately the same time can be computed as follows:

Place a 1ms delay in node two, a 2 millisecond delay in node three and so on.

Resulting discrepancy: Node 1 starts @ \( t = 0.0208 \), node 8 begins @ \( t = 0.01248 + 0.007 = 0.01948 \) sec. yielding a 0.00132 sec. worst case discrepancy between the start of node 1 and node 8. (Note this calculation is for 19200 baud).

Scale factor for baud rate change: 19200/New baud rate.

Example @ 9600 baud, scale factor = 2. Resulting worst case discrepancy = 0.00132*2 = 0.00264 seconds.

This value is then used as the parameter for a "D" command which will be inserted at the beginning of each node’s local program (see figure 3.4.1). When program execution for all nodes is initiated, each individual node will receive the execution command at a known time relative to the other nodes. Since the calculated delay time is the first instruction to be executed, each node will in effect delay program execution by the amount of time equal to that of the network delay. Immediately after the delay command is finished, all of the nodes will be essentially be starting in sync. (Note that this assumes that the node controller on each node has an 11MHz crystal installed as the system clock.)

![Diagram](image)

**Figure 3.4.1:** Location of the synchronization delay command in a node’s local memory.
Immediately following the delay command is the header file. Since this file is the same for all of the nodes on the network and there are no asynchronous operations within it, the nodes will still be in sync when they begin the execution of the user's program. If there is a need to synchronize the nodes after the execution of some asynchronous operation, a method similar to that of one described in the next section can be used.

3.4.2 Using the Global Synchronization Line

Use of the global synchronization line to synchronize the program execution on all of the nodes is by far the superior method of those presented thus far. This method makes use of a single wire which connects to high speed I/O bit number 5 of every node on the network through the network connection itself as shown in figure 3.4.2.

![Diagram of the global synchronization line connections.](image)

**Figure 3.4.2:** Diagram of the global synchronization line connections.

In this method, node 1 serves as the master controller. All other nodes on the network are slaves which await a signal from the master to begin program execution. To achieve this synchronization, all nodes except node 1 execute a wait command (W command) which waits for USR5 (the global synchronization line) to be asserted. Since after the nodes are initialized all user bits are configured as inputs, this command will continue to read zeroes on the input bit, effectively halting program execution until node 1 sets USR5 high. To ensure that all the nodes on the network are in fact waiting for this signal, a delay of 2 millisecond should be included on node 1.
The code to implement this algorithm would be added at the very beginning of the user program files and would appear as follows:

Nodes 2 - 8:

; Wait on the global sync. line.
/W 05H

; Rest of the user program

Node 1:

; Wait for all nodes to hit the global sync line block.
D 2

; Initiate program start.
/B 05H

This method will synchronize the nodes at the beginning of the user program to a much higher degree than the time delayed method. The accuracy is completely dependant on the speed at which the node controller can pole the global synchronization line. If a position update is being processed by a node before reaching the “W 05H” command, that particular node is not guaranteed to have started on the initial edge of the global sync. signal it is therefore possible that it will be out of sync with the rest of the network. For this and several other reasons, a switch is provided on the user interface screen which will start and stop the position updating mechanism. Note that if the nodes are being run independently, it is up to the user to ensure that one of the nodes on the net is holding the global synchronization line high so that individual program execution may continue. Also note that the time delayed method is always inserted by the application program but it will not have an effect on the accuracy of this method.
Chapter 4

4. System Programming

This chapter is intended to serve as a guide to programming the nodes on the network. It describes the architecture of a CN from a programmer's point of view.

4.1 Programmer's Model

Figure 4.1.1 shows an overview of all of the registers present in each controller node. Registers within the CY550 processor may be accessed directly through the various user commands as presented in section 4.2.1. Registers in the MAX505 and the 8255 must be accessed indirectly as described in section 4.2.2.

![Figure 4.1.1: An overview of the registers which are common to all controller nodes.]

As seen in the figure, there are four independent DAC's in the MAX505, each of which may be controlled via the four DAC registers.
As the system has been constructed, DAC A is currently unused, DAC B will hold a value which sets the current in the stepper motor windings as the motor is accelerating or decelerating, DAC C sets the current in the stepper motor windings while the motor is idle, and DAC D dictates the current when the motor as stepping at the maximum rate (see the description of the Rate register). All of the registers on this device are read-only therefore any attempt to read a value from them will produce unpredictable results.

Registers in the 8255 are the control register, port A, port B, and port C registers. The contents of the control register dictates the functionality of the three ports on the 8255. For a complete description of the operation of this device, please refer to the manufacturers data sheets. For the purpose of this discussion, suffice it to say that the three ports can be programmed independently or together as inputs, outputs, or a combination of the two.

In the CY550, there are twelve registers which are visible to the programmer. The first is the position register. This is a twenty four bit register which holds a signed value representing the current position of the stepper motor. Commands are available which can clear and set the value of this register. It also provides a basis for relative and absolute movements. Care must be taken when using this register the signed value can “wrap around” giving an incorrect position value. Wrap around occurs when the absolute value of the number in the position register is larger than 8388608 decimal. If the position is incremented by one more step, the sign of the position will immediately be change from positive to negative or vise versa. Programs in which such a situation might arise should take this into account.

The “Number of Steps” register is used to hold the number of steps specified during a relative move operation. This is also a twenty four bit register.

The “FirstR” register is an eight bit register which contains a pointer in to the first step rate table found in appendix A. The value to which this register points determines the initial step rate of the
stepping motor just before acceleration begins and immediately after deceleration ends.

The “Rate” register is a sixteen bit register which determines the maximum step rate of the motor. Section 2.3 gives a complete description of how to compute the maximum step rate given the contents of this register.

The “Slope” register contains a value which is used to determine the slope of the acceleration and deceleration curves. For more details see section 2.3.

The “Mode” register is used to determine the operating characteristics of the CY550 and should not be changed within a user program. Doing so may result in the loss of communications and/or program data.

The “Delay” register is used to generate time delays between the execution of program instructions. The value in this register translates to a delay given by \( T \) (in milliseconds) = (Decimal equivalent of the contents) * (11 / 12). The fraction comes from the fact that the register is calibrated for a 12 Mhz clock rate rather than the 11Mhz clock that is installed on the CN's.

The “Memory Address” register is synonymous to the program counter in a traditional microprocessor, that is, this register is used to keep a pointer to the next instruction in the program memory which will be executed.

The “Input Address Pointer” register is used to store a pointer into the external address space. When an extended I/O read is performed, the value at the location to which this register points will be read into the I/O register as described in full detail in section 4.2.2.

The “Output Address Pointer” register is used in a similar manner as the Input address Pointer register in that a write operation will place the value in the I/O register at the address pointed to by this register. See section 4.2.2 for complete details.

The “I/O Register” is an eight bit register which is used to store data which has been read from or is destined for external memory space. See section 4.2.2 for complete details.
The “HS I/O Port” register is an eight bit register which contains the status of the USR bits on processor’s I/O port. Bit number seven (USR7) on this port is used by the motor driver circuit to select between half (high) and full (low) step modes. Bit number six (USR6) is reserved for use by the network controller and should not be used in user programs. Bit number five of all of the nodes on the network is tied together to form a global synchronization line. See section 3.4.2 for more details. Bits 0 through 4 are available for general I/O use.

$0000 - $7FFF  RAM
$8000 - $DFFF  Unassigned, available
$E000         DAC A  Unused
$E001         DAC B - Ramp Current
$E002         DAC C - Idle Current
$E003         DAC D - At Speed Current
$E004 - $EFFF  Unused  Reserved
$F000         PIA  Port A
$F001         PIA  Port B
$F002         PIA - Port C
$F003         PIA  Control
$F004 - $FFFF  Unused  Reserved

Figure 4.1.2: The programmers’ memory map.

Figure 4.1.2 shows a summary of the programmer’s memory map which is common to all of the controller nodes. Note that there is space to add additional devices (DAC’s, ADC’s, etc.) to the node in the area from $8000 to $DFFF.

4.2 Programming the Nodes

As mentioned earlier, each of the nodes on the network may be programmed independently of one another through the use of program files. Program files are written on the PC using any type of text editor which produces an ASCII text file. The program file may consist of both comments and commands as described in section 4.2.1. Once a program file is written it may be assigned to a node or
multiple nodes on the network using the Configuration Control Menu described in section 3.2.3.

The application program will do some simple syntax checking on the user files as they are downloaded to the nodes. For best results, the following rules should be followed when writing a program:

- The number of characters on any line should not exceed 256.
- Program files are NOT case sensitive.
- CR and CR-LF file formats are acceptable.
- Only one command per line is allowed.
- If a comment character appears on a program line, the entire line is ignored.
- Observe the proper notation for hexadecimal parameters.
- Both spaces and/or tabs can be used within a file.
- If a command which does not expect a parameter is specified with a parameter present, the parameter will be ignored.
- The first error found in a file is the only error that is reported.
- Labels must occur on a separate line in the program file. Labels are case insensitive. Valid characters are A-Z and 0-9.

Several error messages may result from parsing the user files. All errors that are reported will include the filename and the line number at which the error occurred (where applicable). Note that it is not the focus of this project to develop a compiler and therefore only a few of the many possible errors in user programs have been detected.

- Unresolved Label < labelname >: This error occurs when a label that is used in a branch command has not been defined.
- Duplicate label <labelname > found on line <>: This error occurs when the same label name has been defined twice.
- An invalid label has been found on line <>: This error occurs when invalid characters have been found in a label.
- Invalid label <labelname > in the Y command on line <>: This error occurs when an invalid label is specified in a Y command.
• Illegal label on line <>: This error occurs when a label containing invalid characters has been found.

• An invalid parameter has been specified on line <>: This error occurs when invalid characters are found in a command parameter.

• Command requires 2 parameters. Missing comma? Line <>: This error occurs when a command which requires two parameters has been specified, however only one or no parameters have been supplied.

• Command requires a parameter. Line <>: This error indicates that a command requiring a parameter has been specified without one.

• No valid user command found on line <>: This error indicates that the specified line number contains an illegal command.

• Error opening < filename >: If a program file cannot be located or opened, this error is reported. The filename may be one of the header file, the user file, or the tail file.

• Memory page error assigning label at address <>: This error occurs when a branch instruction cannot reach to specified label due to the addressing constraints. See the following section on looping for more complete details.

4.2.1 Looping

Several looping constructs are available for use within user programs. These commands are the J, L, T, Y and Z commands (see section 4.2.1). Due to the architecture of the CY550, several of these commands have constraints placed on their usage. In the case of the J, L, T, and Z commands, an eight bit address is used internal to the processor to represent a jump offset into the current memory page. The resulting address is generated by concatenating the upper eight bits of the current memory address and the first eight bits of the label address. In cases where the jump instruction and the destination address are on different memory pages, an error message
will be generated at the time the program is downloaded to the node. The only way to alleviate this problem is to rearrange the order of the program instructions in such a way as to satisfy the addressing constraints. Unfortunately, this limits the reach of these branch instructions to memory addresses within the memory page in which the branch instruction resides, therefore branch instructions should be avoided where ever possible. If they must be used, the distance of branching should be kept to a few instructions or designed in such a way as to make use of the sixteen bit Y instruction described in the following paragraphs.

The Z and L commands are designed to create loops in which the number of iterations are held in the counter register. Any number of L loops may reside in a Z loop or, any number of Z loops may reside within one L loop however, loops of the same type may not be nested since there is only one counter register for each type of loop.

The Y and J commands are jump commands which will always jump to the specified label. Note that the Y and J commands may at first glance appear to be the same however the Y command is capable of jumping to any label within the memory space of a processing node where as the J command is limited to an 8 bit offset as described earlier. Therefore, it is highly recommended that the Y command be used in all cases.

Since the T command does not require the use of any processor registers, T loops may be used without nesting restrictions. The only limiting factor is the fact that the T command is also limited to branching within the current memory page. Note that during the program control mode, if a program is stopped and then started again in the middle of a Z or an L loop, when the processor encounters the loop command, it will treat it as though the command had not yet been processed - resetting the counter register to the initial count value.
4.2.2 Extended I/O

Any registers and memory locations which are not located directly in the processor must be accessed through the extended I/O facilities of the processor. As shown in figures 4.2.1 and 4.2.2, reading and writing in these situations requires the use of the pointer registers.

External memory read operations are usually performed in cases where data must be obtained from the ports on the 8255 and so an example of reading from port A is as follows: First, the input address pointer must be set to the address of port A (K 0F000H) which is to be read. Next, an external memory read is performed via the "!" command (!), which places the contents of the port in the I/O register.

A common external memory write operation is one in which the value of a DAC is being set and so as an example, we will set the value of DAC A to decimal 53. First, set the output address pointer register to point to the desired DAC register, in this case the command "M 0E000H" for DAC A. Next, set the I/O register to the desired output value, "# 53". To complete the operation, the data is written to the DAC by executing the "%" command. See section 4.2.1 for a complete summary of user commands.
4.2.1 User Command Summary

This section is intended to serve as a command reference while writing program files. Several of the commands in this section require a parameter which may be specified as either a hexadecimal or as a decimal number. Any time a hexadecimal number is used, it must be preceded by a zero and proceeded by an "H". For example the hex value $FA7 is represented as 0FA7H in a used program. In cases where the parameter is being specified in decimal form, the decimal number serves as the parameter without the need for additional characters.

A, a At current step position.

The "A" command will replace the current contents of the position register with the parameter included with the command. Negative values may specified when using the decimal format by placing a minus sign in front of the parameter. Legal parameter values are 8388607 through -8388608 decimal (0800000H through 07FFFFFFH hexadecimal).

Format: A < 24 bit position value (decimal or hex) >

Examples:
A negative decimal value: A -9874
A positive decimal value: A 9874
A hexadecimal parameter: A 0F7843H
**B,b** Set or clear a high speed I/O bit.

The "B" command is capable of clearing or setting a bit in the high speed I/O port or in the I/O byte register depending on the value of the parameter as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00H - 07H</td>
<td>Set one of the high speed I/O bits.</td>
</tr>
<tr>
<td>10H - 17H</td>
<td>Clear one of the high speed I/O bits.</td>
</tr>
<tr>
<td>20H - 27H</td>
<td>Set one bit in the I/O register.</td>
</tr>
<tr>
<td>30H - 37H</td>
<td>Clear a bit in the I/O register.</td>
</tr>
</tbody>
</table>

Note that in cases where the high speed I/O bits are being used, the "/" modifier may be used in place of the relatively cryptic codes shown above. Also note that other parameters are available but are not recommended for use in this system. See the CY550 data book for more details.

Format: $B < 8$ bit parameter $>$

Examples:

- Set HS I/O bit 5: "$B 5$"
- Clear HS I/O bit 5: "/B 5" or "$B 015H"
- Set I/O register bit 2: "$B 022H"
- Clear I/O register bit 2: "$B 032H"

**C,c** Set continuous step mode.

The "C" command will place the node in a continuous stepping mode. This mode sets up the controller to accelerate from the initial value (see the F command) to the final rate (see the R command) using the current slope value (see the R command) where it will continue to step until a stop motion command (\(^\wedge\)) is executed, at which point the controller will ramp down to the final rate using the slope and stop. Note however that this command merely sets the mode and does not initiate the motion (see the G command). This mode may also be cleared by pulling the inhibit_abort signal on the driver board low. Once the continuous motion has been stopped, the CY550 will return to the incremental positioning mode.

Format: $C$

See also: F, G, R, S, ^
**D, d - Delay**

The “D” command will cause the controller to pause program execution by the number of milliseconds. The parameter may range from 1 to 65535 in decimal (01H to 0FFFFH in hexadecimal). Each increment of the parameter corresponds roughly to a 1 millisecond delay. The actual delay time can be computed as .916667 * parameter value. Note that this command will interfere with live position displays and will produce slightly longer delays if the controller is also stepping when it is executed.

- Format: D < 16bit parameter >
- Examples:
  - Delay for 20.2 milliseconds: D 22 or D 016H
  - Delay for 1 second: D 1091 or D 0443H

**F, f - First step rate**

The “F” command is a pointer into the rate table shown in appendix A. The corresponding entry in the table dictates the initial step rate and the final step rate of the acceleration / deceleration curve. Valid parameter values are 0 through 119.

- Format: F < 8 bit parameter >
- Examples: F 045H or F 69
- See also: C, P, R, S, Section 2.3

**G, g - Start motion (Go)**

The “G” command will initiate a continuous motion (see the “C” command) if the controller is in continuous step mode, and a relative motion (see the “N”, “+”, and “-” commands) if the controller is in a relative stepping mode (set by the “N” command).

- Format: G
- See also: C, N, +, , Section 2.3

**H, h - Seek to home**

The “H” command will cause the motor to run in a counter clockwise direction at a step rate of 1/20 of the first rate value until
the test case is no longer true. The motor will then stop and reverse direction until the match for the parameter is found once again. This serves to create a home function which will use an input bit as a home detection line. Note that other parameters are available but are not recommended for use in this system. See the CY550 data book for more details.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>00H - 07H</td>
<td>Test one HS I/O bit for 1</td>
</tr>
<tr>
<td>010H - 017H</td>
<td>Test one HS I/O bit for 0</td>
</tr>
</tbody>
</table>

Format: \( H < 8 \text{ bit parameter} > \)
Example: \( H \ 03H \)

**J,j - Jump to label**
The "J" command is an 8 bit jump command which will jump to any instruction in the current memory page as identified by a label in the program.

Format: \( J< \_\text{programLabel} > \)
Example: \( J \_\text{EnergizeRelay} \)

See also: L, Z, Section 4.2.1

**K,k - Set the read pointer register**
The "K" command will set the input address pointer to the value of the parameter (16 bits). The physical address loaded in the register can be used to read a byte value into the I/O register as described in section 4.2.

Format: \( K < 16 \text{ bit parameter} > \)
Example: \( K \ 045A3H \)

See also: M, !, %, Section 4.2

**L,l - Loop to label for count**
The "L" command creates a loop which will jump to a label for a specified number of times. Note that L commands may not be nested and that a program which has been stopped may not be continued while inside of an L loop. If the program is continued, the
counter will be reset to the 8 bit count. The count range is from 1 to 255 (0H to 0FFH) and the label must be in the current memory page.

Format: L <8 bit count>, <Label>
Example: L 55, _FirstLoop
See also: J, Z, Section 4.2.1

M, m - Set the write pointer register
The “M” command will set the output address pointer to the value of the parameter (16 bits). This register is used in combination with the I/O register to write a byte value to a location anywhere in the physical address space as described in section 4.2. The parameter range is from 0 to 65536 (0H to 0FFFFH in hexadecimal).

Format: M <16 bit parameter>
Example: M 045A3H
See also: K, !, %, Section 4.1

N, n - Number of steps
The “N” command is used to set the number of steps to take during a relative move. Depending on the direction of the move (see the + and - commands), the new motor position will be the value in the position register immediately before movement begins, + or - the value of the parameter of the N command. The parameter is a 24 bit value which may range from 0 to 16,777,216 (0H to 0FFFFFFH). Note that position wrap around may occur as described in section 4.1.

Format: N <24 bit parameter>
Example: N 034A8H
See also: G, +, -, Section 4.1

P,p - Position for stepping
The “P” command is used to specify an absolute position to step to as noted by the position register. Once executed, the motor will ramp up using the initial rate (see the F command) and the slope value (see the S command) to the final rate (see the R command) where it will continue to step, ramping down to the final position where it will stop. The parameter range is from 0 to
16,777,216 (0H to 0FFFFFFH). Note that this command will initiate the motion.

Format: P <24 bit position parameter>
Example: P 034FF5H
See also: C, F, G, R, S, ^

R, r - Set the maximum step rate.
The "R" command is used to place an upper limit on the step rate of the motor. The maximum step rate of the motor in steps per second is given by the equation:

Max. Step Rate = 11000000 / (12 * parameter).
The parameter is a 16 bit value which may range from 67 to 65535 (043H to 0FFFFH).
Format: R <16 bit parameter>
Example: R 0543H
See also: C, F, P, S, Section 2.3

S, s - Set the acceleration slope
The "S" command is used to set the acceleration during the ramp up state of motion. For a more complete description of this command, see section 2.3. The range of the parameter is 0 to 255 (0H through 0FFH).
Format: S <8 bit acceleration parameter>
Example: S 05FH
See also: C, F, P, R, Section 2.3

T, t - Loop until bit matches value.
The "T" command will create a loop which is continually executed until the parameter test case is satisfied. Note that in cases where the high speed I/O bits are being used, the "/" modifier may be used in place of the relatively cryptic codes shown. Also note that other parameters are available but are not recommended for use in this system. See the CY550 data book for more details.

Parameter   Result
00H to 07H   Test one of the HS I/O bits for 1.
010H to 017H  Test one of the HS I/O bits for 0.
020H to 027H  Test one of the I/O register bits for 1.
030H to 037H  Test one of the I/O register bits for 0.

Format: T < 8 bit parameter >, <label>

Example:

Continue to loop until user bit 2 is high:  T 02H, _reps
Continue to loop until user bit 2 is low:  T 012H, _reps

See Also: _, L, Z, section 4.2.1

V, v - Wait for step to finish
The “V” command will suspend program execution until the current motion has been finished. Note that this command will interfere with the automatic position display system on the PC.

Format: V
See also: ], ^

W, w - Wait for bit to match
The “W” command is used to suspend program execution until the parameter test case is satisfied. Note that this command will interfere with the position updating mechanism.

Parameter  Result
00H to 07H  Wait for HS I/O bit to equal 1.
010H to 017H  Wait for HS I/O bit to equal 0.

The “/” modifier may be used in place of the relatively cryptic codes shown above. Also note that other parameters are available but are not recommended for use in this system. See the CY550 data book for more details.

Format: W < 8 bit parameter >

Examples:

Wait for HS I/O bit 3 to equal 1:  “W 3” or “W 03H”
Wait for HS I/O bit 3 to equal 0:  “W 013H” or “/W 3”

See Also: T
Y, y - Jump to label

The “Y” command will cause program execution to continue at the specified label. Note that this is a full 16 bit jump instruction and can address the full memory space. It is recommended that this command be used in place of the “J” command wherever possible since memory paging is not an issue.

Format: Y <label>
Example: Y _FeedFilm
See also: J, L, Z, Section 4.1

Z, z - Loop to label for count

The “Z” command works the same as the the L command except the L command allows a 16 bit count value to create a larger number of iterations. The count parameter may range from 0 to 65535 (0H to 0FFFFH) and the label must be on the current memory page.

Format: Z <16 bit count>, <label>
Example: Z 04AC3H, _CutHoles
See also: J, Y, L, Section 4.1

+ - Set clockwise direction

The “+” command will set the direction bit on the motor driver to move the motor in the clockwise direction.

Format: +
See also: C, -

“-” - Set counterclockwise direction

The “-” command will set the direction bit on the motor driver to move the motor in the counterclockwise direction.

Format: -
See also: C, +
/ - Negate or clear bit values
The “/” character is used more as a modifier to a command than a command in itself. It is used in cases where a HS I/O bit is to be tested for a zero value for one reason or another.
Format: /<command> <HS I/O bit number>
Examples:
Set HS I/O bit 3 to 0: “/B 3”
Wait for HS I/O bit 3 to equal 0: “/W 3” or “/W 03H”
See Also: B, T, and W.

] - Wait for position
The “]” command will cause program execution to be suspended until the contents of the position register match the parameter value. The parameter may range from 0 to 16,777,216 (0H to 0FFFFFFH).
Format: ] 045FA1H
See also: V, ^

^ - Stop motion
The “^” command will stop any type of motion (continuous, relative or absolute). Upon executing this command, the node controller will go into a ramp-down mode until motion is stopped. The same effect may be achieved by pulling the Inhibit/Abort signal on the motor driver board low.
Format: ^
See also: C, F, P, R, S, V, Section 2.3

# - Set the value of the I/O byte register
The “#” command will place the specified 8 bit parameter value into the I/O byte register. For a complete explanation of I/O operations, see section 4.1. The parameter may range from 0 to 255 (0H to 0FFH).
Format # <8 bit parameter value>
See also: K, M, !, %, Section 4.1
! - Read from memory
The "!" command will read the contents of the memory location as dictated by the Input address pointer register, and place the result in the I/O byte register.
Format: !
See also: K, M, %, #, Section 4.1

% - Write to memory
The "%" command will place the contents of the I/O byte register into the memory location dictated by the Output address pointer register.
Format: %
See also: K, M, !, #, Section 4.1

; - Comment character
The ";" character is used to indicate lines in a program file which are to be ignored by the parser. Any line containing this character, no matter what its position in the line, is considered to be a comment line.
Format: ;
Examples:
   D 450 ; The D command will be ignored.
   ; This is a comment.

_ - Label character
The "_" character is used to mark the beginning of a label name in a user program. Letters and numbers are all valid characters which may be used in a label name. A label name is ended when a space, tab, or carriage return character is found and is case insensitive.
Format: _<labelname>
See Also: J, T, Y, and Z.
Examples:
   _EnergizeRelay
   Z 556, _PulseLoop
4.2.2 System Command Summary

In addition to the commands listed in the previous section, there are several others which are used by the application for network administration purposes. The commands in this category are broken down into two types: network controller commands and CY550 commands. A brief summary of these commands and their usage in the system follows.

The “S” command is the first of the network controller commands and is used for two purposes. The first is to test the physical network connections and the second is to determine the physical address of the nodes on the network. When an S command is issued, each of the nodes receiving the command will add its address onto the end of the command string. When the command returns to the PC, it contains a history of all of the nodes which it has visited. If two nodes on the network have the same address or one of the nodes has an address of $00, the S command will be blocked by that node with the redundant address thus generating a no nodes found error.

The “I” command is a command which may be used to initialize individual network controllers or the CY550’s. In either case, the values of all internal registers will be initialized to their power up default values.

The “E” command informs the CY550 to which the command is directed that the commands that follow should be entered into local memory. The CY550 will continue to store commands until a “Q” command is received.

The “O” command is used to set the operating mode of the CY550. For more information see section 2.1.

The “? P” command is used to query a node for the contents of the position register. Upon receiving this message, the CY550 will generate a return message of the format P=+XXXXXXXX. This command forms the basis for the position updating mechanism.
The "? R" command is similar to the "? P" command except that in this case, the contents of the Rate register will be returned in a message.

Finally, when an "X" command is issued to a controller it will begin execution of its stored program from the address pointed to by the memory address register. Execution from local memory will continue on the node until a "0" command is found or until the application software issues a "^" command (see section 4.2.1).

4.2.3 Sample Programs

The first example program shown here is one which demonstrates the use of the three DAC’s, a simple motion profile, and two HS I/O lines for synchronization to draw a simple rectangle on an XY plotter. We assume that motor which moves the pen in the X direction has been connected to a controller with an alias name "Xmotor" and that the motor which moves the pen in the Y direction has been connected to a controller with an alias name "Ymotor".

In order for the system to function properly, the two synchronization lines are going to be needed. One so that the Xmotor can send signals to the Ymotor and another for the opposite direction. For this purpose a wire is used to connect the USR 0 HS I/O lines of the two controllers together. Similarly, a wire is used to connect the USR 1 lines together to produce the wiring diagram shown in figure 4.2.3.

![Diagram](image)

Figure 4.2.3: I/O connections for the XY plotter example.

For the sake of simplicity, we will assume that the pen is positioned at the upper left hand corner of the rectangle we wish to draw (see figure 4.2.4).
Figure 4.2.4: Resulting figure from the XY plotter example.

The commands to perform this task are as follows (note that comments cannot be included on the same line as the commands however, in the interest of space it has been done here):

**Xmotor**

```
/B 0 ; Clear USR 0.
; Set the DACs to predetermined values.
M 0E001 ; DAC B ramp current.
# 0A4H ; DAC value.
% ; Send it out.
M 0E002 ; DAC C idle current.
# 045H ; DAC value.
% ; Send it out.
M 0E003 ; DAC D At speed.
# 023H ; DAC value.
; Set the motion parameters.
F 1 ; First rate. 15 s/s
S 200 ; Slope.
A 0 ; Reset the position reg.
+ ; Set the CW dir.
N 1234 ; Set the distance.
G ; Start the motion.
V ; Wait for completion.
B 0 ; Signal the Ymotor.
W 1 ; Wait for the Y motor.
- ; Set the CCW dir.
N 1234 ; Set the distance.
G ; Move the motor.
V ; Wait for completion.
/B 0 ; Signal Ymotor.
```

**Ymotor**

```
/B 1 ; Clear USR 1.
; Set the DACs.
M 0E001 ; DAC B - ramp current.
# 0A4H ; DAC value.
% ; Send it out.
M 0E002 ; DAC C idle current.
# 045H ; DAC value.
% ; Send it out.
M 0E003 ; DAC D At speed.
# 023H ; DAC value.
; Set the motion parameters.
F 1 ; First Rate 15 s/s
S 200 ; Slope.
A 0 ; Reset the pos. reg.
; Set the CCW dir.
N 250 ; Set the distance.
W 0 ; Wait for X to finish.
G ; Move the motor.
V ; Wait for completion.
B 1 ; Signal completion.
/W 0 ; Wait for X motor.
+ ; Set the CW dir.
N 250 ; Set the distance.
G ; Move the motor.
V ; Wait for completion.
```
Notice that the Ymotor's program must contain a V command at the end to ensure that the motion has completed before the tail file is executed. Otherwise, the DAC values would be set to zero before the motion had finished. The synchronization lines setup here demonstrate a simple means of synchronizing two motors using the digital I/O ports.

The second example program shown here is a simple program which will blink an LED thirty times with 200 ms on and 200 ms off. It demonstrates the use of labels and the looping construct formed by the L command. For simplicity, we can assume that the LED has been connected to HS I/O bit number 3 and is turned on when that bit is set to a one. The program file to accomplish this is shown below.

```plaintext
; **** Place a label for looping.
_loopLabel
;
; **** Turn off the LED.
/B 3
;
; **** Wait for 200 milliseconds.
D 218
;
; **** Turn the LED on.
B 3
;
; **** Wait for 200 milliseconds.
D 218
;
; **** Set up the loop. It should only be done 29 times since the loop has executed once already.
L 29, _loopLabel
;
; **** Shut the LED off before leaving.
/B 3
```

### 4.3 Pre-programmed Commands

In addition to the program file assigned to a given node, a delay command, a header file, and a tail file is downloaded to each of the nodes as shown in figure 4.3.1. These header and tail files are
the same for every node on the network, even those who do not have an associated user program. A complete listing of these files can be seen in appendix C. Although it is not recommended, these files may be modified to contain any program commands which must be executed by every node on the network. This can be done by saving a copy of the original files "header.stp" and "tail.stp" under another name and adding the new commands to the new files. Note that the names of these files should not be changed and that they should be contained in the same directory on the PC as the application program itself.

![Diagram](image)

**Figure 4.3.1:** Loading order of program files in the processors local memory.

The header file performs three major functions - initialize the DAC values to zero, initialize some of the processor registers, and to create an initial "safe" motion profile. In the event that the user's program does not set these parameters, the header file will have already set them to safe values.

The tail file performs some of the same basic functionality - set the DAC values to zero so that no current will be left in the motor windings once the users program has stopped. The tail file will also issue a stop all motion command followed by a command which indicates the end of the internal program.
5. Performance

In order to measure the accuracy of the synchronization mechanisms, a method of visualizing the results needed to be devised. An X-Y table would have been the best method however, due to cost and general availability, this option was eliminated. The next best method was to mount stepper motors to a toy Etch A Sketch® made by Ohio Art. This drawing toy has two knobs to which an underside mounted “pen” is connected. By turning the knobs, the pen will draw lines on a screen in the same manner as an X-Y plotter draws lines on paper. Although not very accurate, this device was used to verify that the two motors did indeed start at the same time. When a diagonal line was drawn, a flat side would have resulted if one of the motors began stepping before the other.

Based on the discussion in the synchronization section of this document, we expect to see a “jog” in the line when we use the timing synchronization method and a perfectly straight line when using the hardware method. After performing both of these tests using the same motion profile in both methods, there was indeed a jog in the timing synchronization method and a straight line when the hardware synchronization method was used. This jog in the straight line became more pronounced as the number of nodes between the X axis motor controller and the Y axis motor controller increased, as predicted by the calculations.

When a line was drawn which required one motor to move faster than another, difficulties are encountered in producing a straight line since the acceleration curve of the motion profile is non-linear. The same “jog” will result however, if the timing method of synchronization is used.
6. Conclusion

At the outset of this project, several objectives were set forth:

1. To meet the need in the development community for a development/test bed system which would aid in the design and testing of stepper motor based electro - mechanical subassemblies.

2. This system should be designed so as to require a minimum programming background to perform relatively complex operations.

Throughout the design and implementation stages of both hardware and software, these two objectives stood as strict requirements for the system. This chapter serves to take a brief look at the general success of the project and how well these goals have been met.

The first objective of the system is most certainly open to interpretation and is not a hard set milestone. In any event, the system does indeed provide the basic functionality needed to exercise stepper motor based devices. Together with the I/O that is provided and the ability to add additional devices to the system, a large number of applications can be realized.

The second goal of the system raises the question of exactly what does the expression “minimum programming background” mean and how does it apply to this particular application? As applied to this system, this implies a certain degree of user friendliness and perhaps even more importantly, a very short learning curve. In order to achieve this goal, the system was designed to run as a Windows application program with the idea that a large portion of the development community is already familiar with the look and feel of these types of programs thus creating a certain degree of immediate comfort with the system. Approaching this goal from yet another direction, the command set implemented
on the CN's was also chosen for its relatively straightforward instruction set and simple execution methodologies thus allowing relatively complex operations to be performed with a minimum number of instructions.

Perhaps the best measure of the success of this project is the amount of interest and enthusiasm which has been generated as a result of demonstrations at Eastman Kodak. As of this writing, several immediate and long term uses for this project are being pursued.

6.1 Possible Improvements

Throughout the course of designing this system, there were several issues, improvements, and ideas that had to be omitted in some cases in the interest of time and some others were simply not possible with the given processor and architecture. This section briefly explores some of these ideas and outlines many methods for making improvements.

With the introduction of the CY550 and the CY233 in the system, much freedom was sacrificed. While the use of the two devices seems to have drastically reduced the amount of design time needed to produce a workable system, the system was immediately at the mercy of the operating and programming restrictions built into the devices. For instance, many of the user commands interfere with the operation of the mechanism which queries the nodes for position and step rate. This is clearly a problem in this application and several "workarounds" had to be performed to allow for this. Additionally, the internal architecture of the CY550 places many constraints on the usefulness of many of the branching commands, in many cases rendering them useless. Clearly this is a problem which will be evident throughout the lifetime of this system.

Since these two devices do indeed form the operating core of the system and to remedy the problems mentioned above would mean replacing them, a true fix would require almost a complete redesign of the system. Given the time frame of the project, the
decision to use these two devices was a trade-off which made the existing system possible in such a short period of time.

In light of the fact that the architecture cannot easily be changed, there are many other improvements which could be made to the existing system that would make it a more robust tool. For instance, the current serial communications error message asks the user to reset all of the nodes on the network when an error is detected. This could become rather inconvenient when there are eight nodes present in the system. Several of these communications errors could be narrowed down to a particular node or group of nodes on the network. Once the erroneous nodes are determined, a more descriptive message could tell the user which node needs to be reset instead of resetting all of them. This change would be relatively easy to make given the current error detection mechanism. Of course there would still be cases where all nodes would need to be reset.

Another area which could be improved is the method in which the user creates the program files. As the system has been delivered, there is some support for a text editor however there is not one in place. This relatively simple addition would eliminate the need for an additional piece of software to create the files. Taking this modification up to another level, some type of text interpreter could be added onto the front of the system which would expand out the single letter commands into a more readable format. For example the user could type Idle Current = 53, and the commands which are necessary to set the DAC would be generated automatically. The possibilities here are almost infinite.

Looking at the driver circuit, there are several modifications which could be made. For instance a voltage controlled oscillator circuit could be used to vary the chopper driver frequency. Such an addition would allow the system to perform experiments of EMI levels at various frequencies and to more precisely control the current in various size stepper motors. This option was explored in an early phase of the project however the development time for the hardware quickly went beyond the time constraints.
Finally, some method for implementing custom ramp tables into the motion profile would certainly prove useful. Although the existing 120 cover most of the possibilities, there are some cases where a custom table would be a better solution. Such tables can be generated on the existing system through the use of the delay command however the resolution of the table is limited to 1 ms increments and the highest achievable speed is approximately 1000Hz.
Bibliography


## Appendix A

### A. First Step Rate Table

<table>
<thead>
<tr>
<th>F</th>
<th>Steps/second</th>
<th>R</th>
<th>F</th>
<th>Steps/second</th>
<th>R</th>
<th>F</th>
<th>Steps/second</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>14</td>
<td>65576</td>
<td>31</td>
<td>1833</td>
<td>500</td>
<td>62</td>
<td>2642</td>
<td>347</td>
</tr>
<tr>
<td>1</td>
<td>92</td>
<td>10000</td>
<td>32</td>
<td>1867</td>
<td>491</td>
<td>63</td>
<td>2665</td>
<td>344</td>
</tr>
<tr>
<td>2</td>
<td>183</td>
<td>5000</td>
<td>33</td>
<td>1898</td>
<td>483</td>
<td>64</td>
<td>2688</td>
<td>341</td>
</tr>
<tr>
<td>3</td>
<td>275</td>
<td>3333</td>
<td>34</td>
<td>1930</td>
<td>475</td>
<td>65</td>
<td>2712</td>
<td>338</td>
</tr>
<tr>
<td>4</td>
<td>367</td>
<td>2500</td>
<td>35</td>
<td>1959</td>
<td>468</td>
<td>66</td>
<td>2736</td>
<td>335</td>
</tr>
<tr>
<td>5</td>
<td>458</td>
<td>2000</td>
<td>36</td>
<td>1988</td>
<td>461</td>
<td>67</td>
<td>2761</td>
<td>332</td>
</tr>
<tr>
<td>6</td>
<td>587</td>
<td>1562</td>
<td>37</td>
<td>2019</td>
<td>454</td>
<td>68</td>
<td>2786</td>
<td>329</td>
</tr>
<tr>
<td>7</td>
<td>687</td>
<td>1334</td>
<td>38</td>
<td>2046</td>
<td>448</td>
<td>69</td>
<td>2812</td>
<td>326</td>
</tr>
<tr>
<td>8</td>
<td>773</td>
<td>1186</td>
<td>39</td>
<td>2074</td>
<td>442</td>
<td>70</td>
<td>2838</td>
<td>323</td>
</tr>
<tr>
<td>9</td>
<td>849</td>
<td>1080</td>
<td>40</td>
<td>2102</td>
<td>436</td>
<td>71</td>
<td>2865</td>
<td>320</td>
</tr>
<tr>
<td>10</td>
<td>919</td>
<td>998</td>
<td>41</td>
<td>2132</td>
<td>430</td>
<td>72</td>
<td>2892</td>
<td>317</td>
</tr>
<tr>
<td>11</td>
<td>982</td>
<td>933</td>
<td>42</td>
<td>2157</td>
<td>425</td>
<td>73</td>
<td>2919</td>
<td>314</td>
</tr>
<tr>
<td>12</td>
<td>1043</td>
<td>879</td>
<td>43</td>
<td>2183</td>
<td>420</td>
<td>74</td>
<td>2947</td>
<td>311</td>
</tr>
<tr>
<td>13</td>
<td>1099</td>
<td>834</td>
<td>44</td>
<td>2209</td>
<td>415</td>
<td>75</td>
<td>2976</td>
<td>308</td>
</tr>
<tr>
<td>14</td>
<td>1153</td>
<td>795</td>
<td>45</td>
<td>2236</td>
<td>410</td>
<td>76</td>
<td>3005</td>
<td>305</td>
</tr>
<tr>
<td>15</td>
<td>1205</td>
<td>761</td>
<td>46</td>
<td>2263</td>
<td>405</td>
<td>77</td>
<td>3035</td>
<td>302</td>
</tr>
<tr>
<td>16</td>
<td>1254</td>
<td>731</td>
<td>47</td>
<td>2292</td>
<td>400</td>
<td>78</td>
<td>3066</td>
<td>299</td>
</tr>
<tr>
<td>17</td>
<td>1300</td>
<td>705</td>
<td>48</td>
<td>2315</td>
<td>396</td>
<td>79</td>
<td>3097</td>
<td>296</td>
</tr>
<tr>
<td>18</td>
<td>1346</td>
<td>681</td>
<td>49</td>
<td>2338</td>
<td>392</td>
<td>80</td>
<td>3129</td>
<td>293</td>
</tr>
<tr>
<td>19</td>
<td>1389</td>
<td>660</td>
<td>50</td>
<td>2363</td>
<td>388</td>
<td>81</td>
<td>3161</td>
<td>290</td>
</tr>
<tr>
<td>20</td>
<td>1432</td>
<td>640</td>
<td>51</td>
<td>2387</td>
<td>384</td>
<td>82</td>
<td>3194</td>
<td>287</td>
</tr>
<tr>
<td>21</td>
<td>1474</td>
<td>622</td>
<td>52</td>
<td>2412</td>
<td>380</td>
<td>83</td>
<td>3228</td>
<td>284</td>
</tr>
<tr>
<td>22</td>
<td>1513</td>
<td>606</td>
<td>53</td>
<td>2438</td>
<td>376</td>
<td>84</td>
<td>3262</td>
<td>281</td>
</tr>
<tr>
<td>23</td>
<td>1551</td>
<td>591</td>
<td>54</td>
<td>2464</td>
<td>372</td>
<td>85</td>
<td>3297</td>
<td>278</td>
</tr>
<tr>
<td>24</td>
<td>1589</td>
<td>577</td>
<td>55</td>
<td>2491</td>
<td>368</td>
<td>86</td>
<td>3333</td>
<td>275</td>
</tr>
<tr>
<td>25</td>
<td>1625</td>
<td>564</td>
<td>56</td>
<td>2511</td>
<td>365</td>
<td>87</td>
<td>3370</td>
<td>272</td>
</tr>
<tr>
<td>26</td>
<td>1661</td>
<td>552</td>
<td>57</td>
<td>2532</td>
<td>362</td>
<td>88</td>
<td>3408</td>
<td>269</td>
</tr>
<tr>
<td>27</td>
<td>1698</td>
<td>540</td>
<td>58</td>
<td>2553</td>
<td>359</td>
<td>89</td>
<td>3446</td>
<td>266</td>
</tr>
<tr>
<td>28</td>
<td>1733</td>
<td>529</td>
<td>59</td>
<td>2575</td>
<td>356</td>
<td>90</td>
<td>3485</td>
<td>263</td>
</tr>
<tr>
<td>29</td>
<td>1766</td>
<td>519</td>
<td>60</td>
<td>2597</td>
<td>353</td>
<td>91</td>
<td>3526</td>
<td>260</td>
</tr>
<tr>
<td>30</td>
<td>1801</td>
<td>509</td>
<td>61</td>
<td>2619</td>
<td>350</td>
<td>92</td>
<td>3567</td>
<td>257</td>
</tr>
<tr>
<td>F</td>
<td>Steps/second</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>--------------</td>
<td>-----</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>93</td>
<td>3609</td>
<td>254</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>94</td>
<td>3652</td>
<td>251</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>95</td>
<td>3696</td>
<td>248</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>96</td>
<td>3741</td>
<td>245</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>97</td>
<td>3788</td>
<td>242</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>3835</td>
<td>239</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>99</td>
<td>3884</td>
<td>236</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>3917</td>
<td>234</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>3951</td>
<td>232</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>3986</td>
<td>230</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>4020</td>
<td>228</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>4056</td>
<td>226</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>4092</td>
<td>224</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>4129</td>
<td>222</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>107</td>
<td>4167</td>
<td>220</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>4205</td>
<td>218</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>109</td>
<td>4244</td>
<td>216</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>4283</td>
<td>214</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>4324</td>
<td>212</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>4365</td>
<td>210</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>113</td>
<td>4407</td>
<td>208</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>114</td>
<td>4428</td>
<td>207</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>115</td>
<td>4450</td>
<td>206</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>116</td>
<td>4472</td>
<td>205</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>117</td>
<td>4493</td>
<td>204</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>118</td>
<td>4516</td>
<td>203</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>119</td>
<td>4538</td>
<td>202</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
B. Jumper Settings
Note that valid addresses are $01$ through $FF$. $00$ is used by the application software to determine network addresses. No two controllers on the network should have the same physical address.
C. Header and Tail Files

Header File

; First, set the DAC values to 0.
; ****************** DAC B
   M 0E001H
   # 0
   %
; ****************** DAC C
   M 0E002H
   %
; ****************** DAC D
   M 0E003H
   %
; ****************** DAC A
   M 0E000H
   %
; ****************** DAC's are set.
; ****************** Set some initial profile parameters.
   R 65535
   S 1
   F 1
   A 0
Tail File

; First, set the DAC values to 0.
; ****************** DAC B
   M 0E001H
     # 0
     %
; ****************** DAC C
   M 0E002H
     %
; ****************** DAC D
   M 0E003H
     %
; ****************** DAC A
   M 0E000H
     %
; ****************** DAC's are set.
; ****************** Stop command.
^ 0
D. Cable Connections

Appendix D
E. Schematic Diagrams
## Appendix F

### F. Quick Command Reference

<table>
<thead>
<tr>
<th>Command, Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, 24 bit Position</td>
<td>Sets the current step position.</td>
</tr>
<tr>
<td>B, Bit #</td>
<td>Sets or clears a single bit of the high speed I/O port.</td>
</tr>
<tr>
<td>C</td>
<td>Set continuous step mode.</td>
</tr>
<tr>
<td>D, Delay value (16 bits)</td>
<td>Delay for the specified number of milliseconds.</td>
</tr>
<tr>
<td>F, Rate</td>
<td>Specifies one of 120 predefined ramp tables.</td>
</tr>
<tr>
<td>G</td>
<td>Initiate motion in the relative step mode.</td>
</tr>
<tr>
<td>H, Bit</td>
<td>Seek home position using the specified signal.</td>
</tr>
<tr>
<td>J, Addr</td>
<td>Jump to the specified memory address.</td>
</tr>
<tr>
<td>K, 16 bit address</td>
<td>Set the input address pointer for extended I/O reads.</td>
</tr>
<tr>
<td>L, count, Addr</td>
<td>Loop to Addr the specified number of times.</td>
</tr>
<tr>
<td>M, 16 bit address</td>
<td>Set the output address pointer for extended I/O writes.</td>
</tr>
<tr>
<td>N, 24 bit number</td>
<td>Set the number of steps for relative moves.</td>
</tr>
<tr>
<td>P, 24 bit position</td>
<td>Step to the specified absolute position.</td>
</tr>
<tr>
<td>R, 16 bit rate</td>
<td>Slew step rate.</td>
</tr>
<tr>
<td>S, Slope</td>
<td>Acceleration slope value.</td>
</tr>
</tbody>
</table>
T, Bit #, Address
Loop to the specified address until Bit matches value.
V
Wait for completion of current motion if any.
W Bit
Wait for specified bit to match value.
Y 16 bit address
Set external memory address pointer.
Z 16 bit count, Addr
Loop to address the specified number of counts.
+
Set CW direction for relative motions.
-
Set CCW direction for relative motions.
/
Negate the prefix used with bit commands.
], 24 bit position
Wait for the specified absolute position.
^ Stop only the current motion.
#, value
Set the value of the I/O byte register.
!
Extended I/O read.
%
Extended I/O write.
;
Comment character.
-
Label character.