To the Graduate Council:

I am submitting herewith a thesis written by James Richard Franks entitled "Open Architecture CNC for a 3 Axis Milling Machine". I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Mechanical Engineering.

William R. Hamel, Major Professor

We have read this thesis and
recommend its acceptance:

Rajiv V. Dubey

Spivey S. Douglass

Accepted for the Council:

Associate Vice Chancellor and
Dean of The Graduate School

OPEN ARCHITECTURE CNC FOR A 3 AXIS MILLING MACHINE

A Thesis
Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

James Richard Franks
August 1998

Copyright ©, 1998 by James Richard Franks
All rights reserved.

This thesis is dedicated to my lovely wife, Becky, for her patience in reading and editing and to our parents Richard (Dad), Nancy (Mom), Joe (Sir), and Jerri for their support of graduate degrees and our marriage.

Acknowledgments

I would like to thank my wonderful wife Becky for her patience during the final semester of work. Late nights and a tired husband made the spring of 1998 a difficult time. She was always patient with my schedule and my stress level. Thank you for your constant support. I love you!

I would also thank my advisor Dr. Bill Hamel for making graduate school possible. Under Dr. Hamel's guidance I have been able to work on several challenging projects with outstanding resources and people. Dr. Hamel has allowed me grow in areas of engineering I was totally unfamiliar with before graduate school. He and his wife, Libby, have been kind enough to be involved with ME graduate students and their families on a personal level.

Dr. Rajiv Dubey was a very patient teacher especially during my first semester of graduate school. My seemingly endless and sometimes meaninglessless questions would never to slow him from explaining the concept, "just one more time."

Dr. Spivey S. Douglass has been an encouragement since the day I meet him. Spivey was a constant breath of fresh air and sound advice. His dedication to me, the Bridgeport conversion, and the University of Tennessee are sincerely appreciated. Whatever Mississippi State University taught you and Edwin I. Griggs about being great professors, they need to teach it to the rest of academia.

Smiley Clapp and Garland Smith of the Center for Industrial Service (CIS) played an integral role in completing the Bridgeport conversion. CIS paid my salary during last year of graduate school thereby making my Master's degree financially possible. I am truly grateful. Garland Smith was always optimistic about the open architecture control concept and losing him was a great loss to the University of Tennessee and CIS.

Todd Feldhaus and Ralph Kropp of Manufacturing Data Systems Inc. played an integral role in converting the Bridgeport to open architecture control. Ralph was able to provide the University of Tennessee with OpenCNC software at a minimal cost. Todd was kind enough to explain the world of machine tool electrical hardware to a young mechanical engineer. Todd would patiently answer my endless streams of phone calls and questions. Thanks Todd!

I would like to thank Jeff Parrott, Paul Demint, Ted Williams, and Van Graves of Oak Ridge National Laboratory. Jeff, Paul, and Ted were instrumental in acquainting me with finer points electrical engineering and electrical hardware. They would freely make time in their busy schedules to answer electrical hardware questions. Van Graves would answer software questions and drop by from time to time just to see how the project was coming

along.

I would like to thank my classmates and friends from the Robotics and Electromechanical Systems Laboratory: Steve Everett, Stacey Mitchell, David Lennon, Chris Widner, Scott Marland, Amanda Barnette, and Yasinobu Isoda. Steve has been my friend and one of my best teachers in graduate school. The lab is really going to miss you, Steve. Stacey Mitchell has been my constant "goto guy" regarding computer issues and a true pal through out my graduate career. David and I sprinted to finish our theses by May of 1998. We made it, Dave! Chris has helped me with coursework, Latex problems, and general questions about finishing a thesis. Thanks Chris!

Danny Graham, Gary Hatmaker, and Steve Hunley were very helpful in explaining how a machinist views CNC control. Gary was tremendously helpful in giving advice on rewiring the Bridgeport. Danny was a good friend and gave sound advice regarding machining setup and practice. Thanks guys!

My family has been a constant line of support through out my graduate career. Mom, Dad, Jerri, and Sir were very supportive of our marriage and academic careers. Their prayers made this year tolerable and successful.

Finally, I would like to thank God for making this whole experience possible. God allowed a rookie engineer to successfully cut out the control of working machine tool and refit it in approximately 4 months. God provided me with the people and the resources at just the right time. In my weakness I found your strength!

Abstract

Open architecture based computer numerical control (CNC) is an emerging technology in the machine tool industry. Traditionally, machine tool manufacturers have provided proprietary CNC controllers in their products. These proprietary controllers tend to be expensive to repair and upgrade. The machine tool owner was forced to rely upon the machine tool manufacturer for replacement parts and maintenance. Open architecture CNC uses commercially available electrical and personal computer (PC) hardware to implement machine tool control. Motion control algorithms are performed in software on a standard PC rather than a hardwired motion control card. This thesis discusses the background of CNC and the conversion of a 3 axis milling machine to a commercially available open architecture CNC. Another important aspect of machine tool control is the process of designing a part in a computer aided design(CAD) software followed by using a computer aided manufacturing (CAM) software to produce a part program. Two commercially available PC based CAD and CAM packages were examined.

Contents

1 Introduction 1

2 CNC - History and Trends 3
- 2.1 Numerical Control (NC) . . . 3
- 2.2 Computer Numerical Control (CNC) . . . 8
- 2.3 Open Architecture CNC . . . 9
  - 2.3.1 GM Powertrain (Open Modular Architecture Control) . . . 9
  - 2.3.2 MDSI . . . 13
- 2.4 Future Speculation . . . 14

3 Milling Machine Control 15
- 3.1 Single Axis Motion Control . . . 18
  - 3.1.1 Encoders . . . 20
  - 3.1.2 Amplifier Cards and Tachometers . . . 21
  - 3.1.3 Transmission . . . 22
  - 3.1.4 DC Servo Motors . . . 23
  - 3.1.5 System Modeling . . . 25
- 3.2 Coordinated Motion Control . . . 27
  - 3.2.1 Point To Point Machining . . . 27
  - 3.2.2 Continuous Path Machining . . . 29
- 3.3 Spindle Drives . . . 31
- 3.4 Digital Inputs and Outputs . . . 31
  - 3.4.1 Digital Inputs . . . 32
  - 3.4.2 Digital Outputs . . . 33

4 Conversion to Open Architecture Control 34
- 4.1 Chapter Format . . . 36
- 4.2 Axis Definitions . . . 36
- 4.3 Axis Motion Control . . . 37
- 4.4 Spindle Control . . . 39
- 4.5 Hardware Configuration . . . 40
  - 4.5.1 Digital Inputs and Outputs . . . 44
  - 4.5.2 Emergency Stop Circuit . . . 52

4.5.3 Encoder Industry Pack . . . 55
4.5.4 DAC Industry Pack . . . 58
4.6 Software Configuration . . . 62
4.6.1 Operating Systems . . . 62
4.6.2 Network Configuration . . . 63
4.6.3 ISaGRAF^{TM} . . . 64
4.6.4 /home/bridge . . . 65
4.6.5 /usr/mdsi/2.0/machines/bridge . . . 69

5 PC Based Computer Aided Manufacturing **72**
5.1 SolidWorks97Plus . . . 73
5.2 SURFCAM 7.0 . . . 80
5.2.1 Opening a SolidWorks97Plus file . . . 80
5.2.2 Modifying Surface Arrows . . . 84
5.2.3 Cutting . . . 84
5.2.4 Drilling . . . 84
5.2.5 NC Operations Manager . . . 89
5.2.6 Post Processing . . . 89

6 Conclusions and Future Work **92**
6.1 Conclusions . . . 92
6.2 Future Work . . . 93

Bibliography **95**

Appendices **98**

A Operator's Manual **99**
A.1 Control Panel Layouts . . . 99
A.1.1 Miscellaneous Hardware . . . 100
A.1.2 Hardwired Switches . . . 101
A.1.3 Touch Screen Layout . . . 101
A.2 Startup Procedure . . . 103
A.2.1 Turning on the Milling Machine . . . 104
A.2.2 Energizing the Emergency Stop Circuit . . . 105
A.2.3 Syncing the Machine . . . 105
A.2.4 Loading a part program . . . 106
A.2.5 Determining Fixture Offsets . . . 107
A.2.6 Determining Tool Offsets . . . 108
A.2.7 Starting a Part Program . . . 108

B Accounts and Passwords **110**

C QNX Information 111
C.1 Common Commands 111
C.2 Formatting a floppy disk for the QNX Operating System 111
C.3 Mounting & unmounting a QNX floppy disk 112

D sysinit.1 113

E sysinit.2 115

F .profile 117

G startbgp1 118

H dioSymsbgp1 119

I tunebgp1 120

J bgp1.gen 124

K bgp1.men 130

Vita 131

List of Tables

2.1 Tasks of the original contract between Parsons Corporation and the US Air Force 4
2.2 Agreement between Parsons Corporation and MIT's Servo Lab 5

4.1 Axis Numbers 37
4.2 24 VDC Digital Inputs on the Bridgeport V2XT Milling Machine 47
4.3 24 VDC Outputs on the Bridgeport V2XT Milling Machine 50
4.4 110 VAC Outputs on the Bridgeport V2XT Milling Machine 54
4.5 Pin Assignments for the Tech 80 Model 12 Encoder Module 56
4.6 Pin Assignments for the IP-DAC 59
4.7 HB Series Amplifier J1 Terminal Strip Pin Assignments 61
4.8 HB Series Amplifier J2 Terminal Strip Pin Assignments 62
4.9 /etc/config/netmap 64
4.10 Ethernet Cards' Information 64
4.11 runbridge 68

B.1 Accounts and Passwords 110

C.1 Common QNX Commands 111

List of Figures

1.1 Undergraduate student design process using CAD and MSL 1

2.1 Traditional Manufacturing Method 6
2.2 Manufacturing Method with Numerical Control as of 1952 6
2.3 Manufacturing Method with Numerical Control using APT 7
2.4 Manufacturing Method with Numerical Control and CAD 8
2.5 Progression to the Open Modular Control System 10
2.6 Open Modular Architecture Controller 11

3.1 Coordinates of a 3 Axis Milling Machine 16
3.2 CNC Control for a 3 Axis Milling Machine 17
3.3 Single Axis Control Block Diagram 19
3.4 Encoder Direction Determination 21
3.5 Machine Tool Transmission 23
3.6 DC Armature Circuit 25
3.7 Free Body Diagram of Single Axis 25
3.8 Point to Point Machining 28
3.9 Linear Interpolation 30
3.10 Spindle Control Block Diagram 31
3.11 Overtravel Switch Configurations 32

4.1 OpenCNC Control Configuration 35
4.2 Coordinate Definition 38
4.3 Axis Control Block Diagram 39
4.4 Spindle Control Block Diagram 41
4.5 Tech 80 Industry Pack Configuration 43
4.6 Hardware Location 44
4.7 Western Reserve Card 45
4.8 24 VDC Digital Input Wiring Diagram 49
4.9 24 VDC Digital Output Wiring Diagram 51
4.10 110 VAC Digital Output Wiring Diagram 53
4.11 5 VDC Wiring Diagram 54
4.12 Emergency Stop Circuit 55
4.13 Encoder Wiring Diagram 57

4.14 IP-DAC Wiring Diagram 60
4.15 ISaGRAF^{TM} bgp1dio project 66
4.16 ISaGRAF bgp1msg project 67

5.1 Design Process 72
5.2 SolidWorks97Plus Work Screen 74
5.3 SolidWorks97Plus with Profile Sketch 75
5.4 SolidWorks97Plus with Extruded Profile 76
5.5 SolidWorks97Plus with Bearing Hole Sketch 77
5.6 SolidWorks97Plus with Bracket Hole Sketch 78
5.7 SolidWorks97Plus Bearing Bracket without fillets 79
5.8 SolidWorks97Plus Bearing Bracket Model 81
5.9 SolidWorks97Plus File Saving Window 82
5.10 SURFCAM Operator's Window 83
5.11 SURFCAM Surface Arrows 85
5.12 Tool Information Window 86
5.13 Select Tool Window 87
5.14 Program Information Window 88
5.15 NC Operations Manager 90
5.16 Part Programming Editor 91

A.1 Bridgeport Milling Machine with OpenCNC^{TM} control 99
A.2 Miscellaneous Hardware 100
A.3 Hardwire Switches beside the touch screen 101
A.4 Hardwire Switches 102
A.5 OpenCNC Window 103
A.6 MDSI Operator Panel - BGP1 Window 104
A.7 Machine Power Switch and Computer 105
A.8 Home Position 106
A.9 MDSI 1 Program Window 107

Chapter 1

Introduction

This thesis presents the results of a cooperative effort between the University of Tennessee's College of Engineering (COE) and the Center for Industrial Service (CIS). The purpose of the project was two fold: (1) evaluate and demonstrate the details of converting a machine tool control system to a low cost personal computer (PC) based system, and (2) provide a flexible computer aided design environment which can be used to teach undergraduate students modern computer-based manufacturing principles. The tools and processes used to achieve these goals are summarized in Figure 1.1.

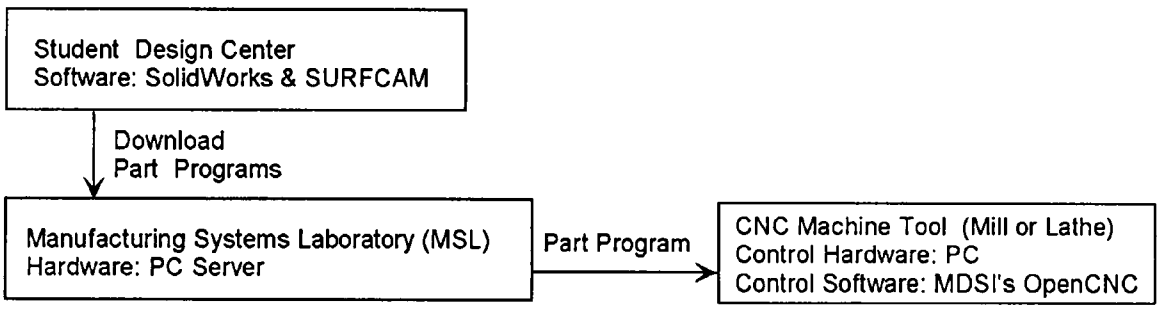


Figure 1.1: Undergraduate student design process using CAD and MSL

The students will design their parts using a CAD package called SolidWorks^{TM} [1] in conjunction with a program called SURFCAM^{TM} [2]. SolidWorks^{TM} is a three dimensional (3D) modeling package that allows the design of multiple parts while viewing how the parts integrate using SolidWorks' simulation tools. After a part has been designed, a computer aided manufacturing (CAM) program, SURFCAM^{TM} simulates the manufacturing of the

part on a machine tool. SURFCAM^{TM} also produces the necessary part program to manufacture the part on a real machine tool. Using an Ethernet connection, part programs are downloaded to the MSL server. The server forwards the part program to the appropriate machine tool.

The machine tools will be controlled using a personal computer (PC) running Manufacturing Data Systems Inc.'s (MDSI) OpenCNC^{TM} machine tool control software. OpenCNC is an example of open architecture machine tool control. Open architecture CNC uses relatively inexpensive, commercially available PC hardware to control various types of machine tools, regardless of the manufacturer. Since the actual control is fundamentally software and not hardware, open architecture control eliminates inflexible proprietary hardware and provides flexible and upgradable control systems for machine tools.

The Center for Industrial Service was founded with a mission, "To help owners and managers of Tennessee industrial firms increase their competitiveness by helping them solve production problems, boost productivity and quality, introduce new technology, and improve training of their employees.[3]" CIS is interested in assisting Tennessee manufacturers with low cost CNC technologies. The milling machine project being reported here is considered an appropriate example of the skills necessary to convert and operate a system with open architecture control.

This thesis covers the history of CNC and specifically the conversion of a Bridgeport 3-axis milling machine to open architecture control using MDSI's OpenCNC^{TM} software. Chapter 2 covers the history of CNC and the development of open architecture CNC. Chapter 3 reviews milling machine control from the servo drives to the control software. Chapter 4 covers the conversion of the Bridgeport milling machine to open architecture control. Chapter 4 was written to provide documentation for the operators and engineers using the Bridgeport mill. It covers the following topics: hardware, wiring, networking, and software programming. Chapter 5 describes the use of PC based computer aided drafting and manufacturing software to aid in engineering design and part program production.

Chapter 2

CNC - History and Trends

2.1 Numerical Control (NC)

The origin of a numerically controlled machine tool began with John T. Parsons' of the Parsons Corporation, Aircraft Division [4]. Parsons had experimented through the 1940's with a discrete-position machining method. The discrete-position machining method could produce two dimensional parts, such as cams, by positioning the blank stock of material under a cutting tool. Production involved cutting a discrete series of holes and then finishing or smoothing the part. The manual procedure would be to mount the part to a table that moved in two directions. Next, the cutting tool would be positioned above the blank stock and then brought down to make the first cut or hole. The stock would be moved to the next position by adjusting the lead screws on the table by means of a hand crank.

Parsons believed this procedure could be automated by the use of a computer. A computer would read the coordinate information via punch cards, and the lead screws would be adjusted automatically. Later, Parsons believed this idea could be expanded so continuous cuts could be made automatically. This type of technology would allow tapered wing sections to be produced economically. Parsons made a formal proposal to the Air Force and was rewarded with a contract. The contract specified tasks to be performed as shown in Table 2.1. Furthermore, the contract stipulated that money would only be advanced as tasks were completed.

John Parsons soon realized his company needed help in order to meet the terms of their

Table 2.1: Tasks of the original contract between Parsons Corporation and the US Air Force

Task #	Task to be completed	Money Allotted
1	Design study of a servomechanism actuated by punch cards.	$50,000
2	Construction and testing of a 18" x 24 " table under working conditions.	$40,000
3	Design of a 3 axis machine with a part capacity of 18" x 24" x 48" able to machine steel and aluminum.	$20,000
4	Construction and testing of the machine tool designed in part 3.	$90,000

agreement with the Air Force. Parsons contacted Dr. Gordon S. Brown at the Massachusetts Institute of Technology's (MIT) Servomechanisms Laboratory (Servo Lab) for assistance in 1949. The Servo Lab in MIT's department of Electrical Engineering was founded in 1940 by Dr. Brown and was used as an outlet for the faculty to conduct research. Dr. Brown taught courses in feedback control throughout the 1930's. With the development of World War II, the Servo Lab became involved in gunfire control, missile guidance control, flight simulators, and digital computers. Dr. Brown was interested in Mr. Parsons' concept of controlling a servo drive with digital inputs.

On July 29, 1949 Parsons Corporation and MIT's Servo Lab reached an agreement to conduct research toward a machine tool controlled by computer inputs. The research outlined in Table 2.2 did not have ties to the contract between Parsons and the US Air Force. The research would be conducted by the staff of the Servo Lab over the period of one year.

Shortly after this agreement was signed, several practical problems were realized. First, after consulting with IBM, the Servo Lab learned that commercially available card readers were not fast enough for the feed rates desired for the cutting tool. The alternatives were to place the information on a punched paper tape or a magnetic tape, and IBM was not interested in aiding the Servo Lab in pursuit of these new avenues of technology. Second, upon further review of the problem, the Servo Lab determined it had underestimated the

Table 2.2: Agreement between Parsons Corporation and MIT's Servo Lab

Phase	Research	Money Allotted
1	Research toward the development of a system to control the axes of a machine tool with the inputs from a digital computer.	$30,000
2	To demonstrate this technology on a machine tool to produce a part with a surface roughness less than 0.001 inches with the machine tool placed correctly with in 0.0004 inches when operating at commercial feed rates.	$80,000

amount of digital signal processing necessary for this project. Third, Parsons was trying to narrow the research by specifying a machine tool for the production of taper aircraft wing sections. The Servo Lab desired to keep the application general enough to be applied to various machine tools. Because of the problems involving the Air Force/Parsons contract, the Parsons/MIT contract was renegotiated. In January 1950 the new project goals were to (1) build a test axis to test the principles involved in controlling one axis, (2) control one axis of a 3 axis mill, and (3) control two axes of a 3 axis mill.

The second year of the project involved designing and testing the various subsystems. Each circuit was designed on paper, built, and tested in order to determine if it would meet the design specifications. An ambitious program of testing vacuum tubes was begun. At the time vacuum tubes were thought to be unreliable. It was believed that vacuum tubes would fail in the early hours of use. The Servo Lab began testing and "preburning" of various types of vacuum tubes in order to determine which tubes were reliable. Although vacuum tubes were a vital piece of technology at the time, the development of solid state circuitry ended their necessity.

By January 1951, the contract between the Air Force, MIT, and the Parsons Corporation was being renegotiated. Parsons Corporation was becoming financially unstable, therefore, the US Air Force sought to a direct relationship between itself and the Servo Lab. Although all parties involved were anxious to keep the project going, the work almost ended early

due to the Air Force's failure to appropriate funds correctly and in a timely fashion.

By March of 1952 a Cincinnati Milling Hydro-Tel three axis mill had been out fitted with numerical control. The computer or director would receive instructions from a paper tape input and convert the digital signal from the computer to an analog signal for a direct input to the mill's servos. The Whirlwind Computer, developed by the Servo Lab in the 1940's as one of the early digital computers, read the paper tape.

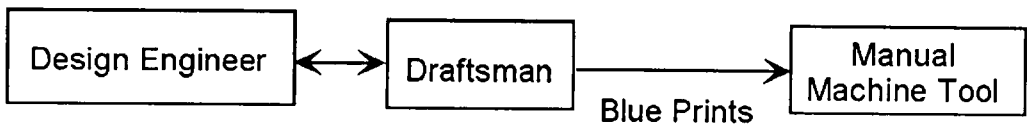


Figure 2.1: Traditional Manufacturing Method

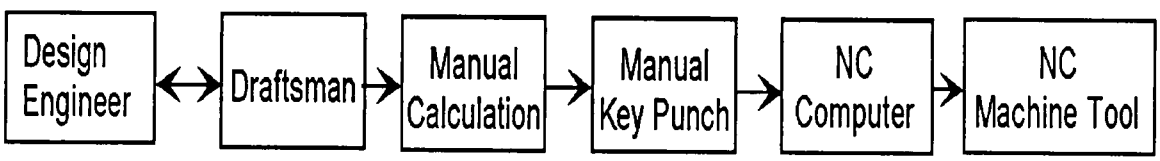


Figure 2.2: Manufacturing Method with Numerical Control as of 1952

The traditional manufacturing method, before numerical control was developed, involved the engineer designing the part and the draftsman producing a set of prints. The prints were sent to the machinist for manual production on a machine tool as shown in Figure 2.1. With numerical control several steps were added to the manufacturing process as shown in Figure 2.2. First, the prints from the draftsman were sent to a clerk using a mechanical calculator to make the necessary calculations to program the numerical control computer. These calculations would then be sent to another clerk to manually punch holes through a paper tape which would provide the input to the numerical control computer. At this point, the system is automated and will function with little human interaction. Obviously, the steps of manually calculating the programming sequence and punching holes in the tape were the most time consuming in the process. The Servo Lab, therefore, began pursuing a method to automate the manual calculation and key punching steps.

The initial work to reduce the amount of manual calculation was done by John Runyon

and Arnold Siegel. Runyon provided the programmer with a vast library of subroutines, which ran on a digital computer, to reduce the number of manual calculations performed with an electromechanical calculator. Runyon had subroutines to calculate tool-center offsets, cut length, cut spacing, and linear curve fits. His efforts reduced the amount of new code written by 75 to 90 percent. Siegel, on the other hand, sought to create a computer language that consisted of words and alphanumeric expressions to run on the Whirlwind Computer.

The fundamental change came from mathematician Douglas Ross. Ross sought to create a computer language using commands that would facilitate more effective communication between humans and computers. Ross explained the fundamentals of the problem by saying, "The language must bridge the gap between the fundamentally incompatible characteristics of the two parties. The human is quick-witted but slow, while the computer is slow-witted, but extremely fast." [4] Ross developed the automatically programmed tool system (APT) by taking a very general approach to the nature of programming. Figure 2.3 shows the new manufacturing method for numerical control with the aid of the APT system. The manual calculation and manual keypunch tasks were replaced by a part programmer who takes the prints and writes the program for the particular part.

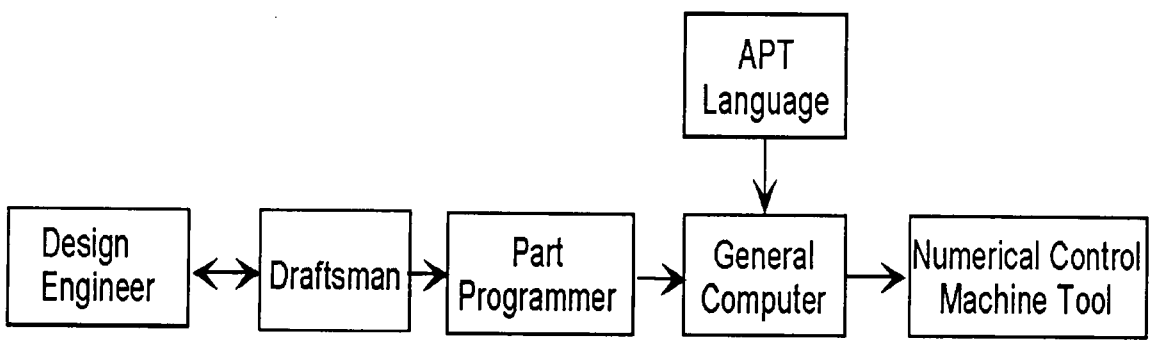


Figure 2.3: Manufacturing Method with Numerical Control using APT

Ross' APT method of programming a machine tool was accepted and maintained by the Aircraft Industries Association, now the Aerospace Industries Association (AIA) until 1962 when the Illinois Institute of Technology Research Institute (IIRTI) became the managing

agency. APT was nominated to become an international standard in 1967 and became one in 1978 by the authority of the International Standards Organization (ISO). Today, APT is the standard NC programming language for machining complex geometries.

The Servo Lab conducted research in 1970's in the area of computer aided design (CAD) of manufactured parts as shown in Figure 2.4. Automated Engineering Design (AED) was an idea to create a direct interface between the design engineer and the computer. The design engineer would design parts using a computer terminal and submit these electronic drawings to the main computer. These drawings would be interpreted by the main computer running the AED program. AED would produce the part programs necessary for part production by a numerically controlled machine tool. This was the last effort of MIT's Servomechanisms Laboratory in the area of numerical control of machine tools.

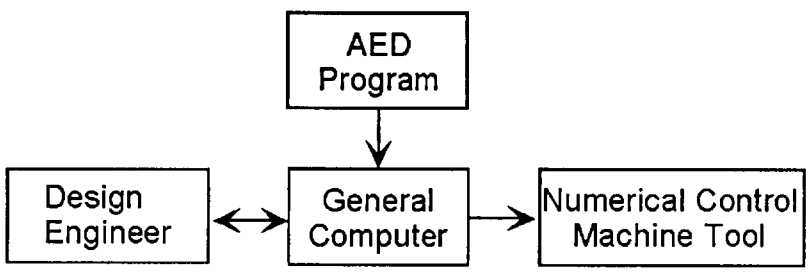


Figure 2.4: Manufacturing Method with Numerical Control and CAD

2.2 Computer Numerical Control (CNC)

With the development of the integrated-circuit computers and cathode ray tubes (CRT), computer numerical control emerged. In contrast to traditional numerical control, computer numerical control (CNC) placed a computer devoted solely to control a specific machine tool. Numerical control required functions such as interpolations, tape reading, and positioning to be done with hard wired electronic circuits. With CNC most of these functions could be performed within the software providing lower costs and greater reliability.

CNC configuration allows several operations to happen at the machine tool. The CRT or monitor allows an operator observe different parameters such as position feed rate, and

spindle speed while the part is being produced. In the event of an error, a warning with corrective action could be flashed to the screen. Traditional NC machine tools required that all instruction be given through a tape reader. With today's modern computer, programs can be downloaded via diskettes/ magnetic tape, or over a network. Once the program reaches the CNC machine tool, editing can still be performed if problems occur. In short CNC made the production process less expensive and more flexible to change.

2.3 Open Architecture CNC

In the years following MIT's work, dozens of machine tool manufacturers began producing their own versions of CNC machine tools. Competition between the manufacturers centered on the number of options they could offer. After a customer purchased a machine tool from manufacturer X, the customer would be monetarily bound to company X for maintenance, repairs, and upgrades to the control system. Although CNC units were significantly more agile than there hardwired NC equivalents, they still lacked the flexibility desired on the modern factory floor. Several companies began exploring how to make CNC more open (free from proprietary restraints), flexible (allow use commercially available products), and economical.

2.3.1 GM Powertrain (Open Modular Architecture Control)

In 1994, the General Motors (GM) Powertrain began producing a series of documents describing their desire for open modular architecture control (OMAC) to be used in the automotive industry [5] [6]. GM Powertrain did not seek production of a rigid design specification for OMAC, but rather to provide the general principles by which future control solutions would be evaluated.

"Open" implied that the control system must use readily available hardware and software to eliminate the problems associated with proprietary components. "Modular" signified the use of some products which are "plug and play" into the control system. GM Powertrain

contended that the use of open modular components would make the control system scalable, economical, and maintainable. With open and modular hardware, the control system could be configured to have enough hardware and software capability to perform a specific ranges of tasks. With modular hardware the control system could be scaled up or down if the task range for the machine tool changed. This would eliminate the scrapping of an entire proprietary control system so that a few new functions could be added. Using commercially available products would allow free market competition to keep product costs low while encouraging constant improvement and support from manufacturers. Finally, a maintainable control system would be stable enough to support a complex manufacturing operation for significant amounts of time while maintenance would be quick and simple to perform.

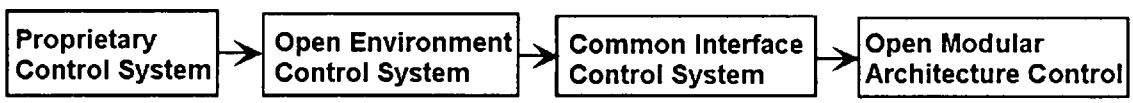


Figure 2.5: Progression to the Open Modular Control System

Since OMAC was defined as a "wish list" for a control system, Figure 2.5 shows the progression from a proprietary control system to OMAC. A proprietary control system is a control system that is completely supplied and supported by the machine tool manufacturer. An open environment control system involves the engineering staff buying off the shelf computer and machine tool hardware and integrating the various components. This integration requires significant effort and expertise from the design staff in various areas of engineering and computer science. The common interface control system involves the use of computer hardware with common application programming interfaces (API). API's would be modularized hardware and software components designed to interface with standard computer hardware and software. These various API components would collectively be able to control the various functions on the machine tool. API's would provide the desired "plug and play" capability for the control system. The final step to OMAC involves putting as much technology into software as opposed to hardware. Instead of having various hardware modules that are integratable into a standard computer, most of the intelligence

would be in software. For example motion control modules are currently available on the market. However, with today's processing speeds it is reasonable for the motion control calculations to be performed on a standard PC computer.

GM Powertrain sought to divide the control needs into parts or modules as shown in Figure 2.6. GM Powertrain desired to build their OMAC around commercially available computer hardware and operating systems. The graphical user interface (GUI) allows an operator to control the machine tool. The motion control of the axes and spindle would be controlled with most of the intelligence in the software rather than the hardware. The network connection would allow the machine tool to communicate with other machines tools, manufacturing controllers, and the engineering design department. The digital inputs and outputs would provide a commercially available interface to monitor machine tool hardware.

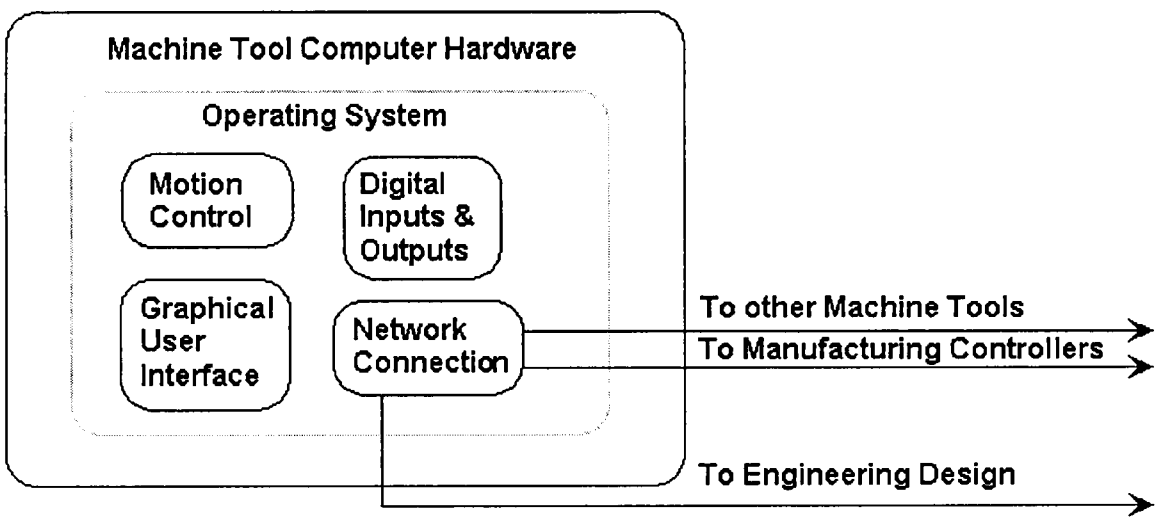


Figure 2.6: Open Modular Architecture Controller

Computer Hardware Platform

GM Powertrain sought to use readily available computer systems in OMAC. Using standard PC hardware, rather than specialized industrial versions, is desirable so that market force can influence cost and improvements. Higher end PC motherboards, such as Intel motherboards, have a mean time between failure (MTBF) of 90,000 hours. This figure is an

impressive value even for hardware specifically designed for industrial purposes. Therefore, PC hardware can be reliable when configured in the appropriate manner.

Debate exists over the use of PC motherboards versus passive backplanes. Common commercial backplanes such as Industry Standard Architecture (ISA), VERSA Module Eurocard (VME), and Peripheral Component Interconnect (PCI) connections were considered. Although passive backplanes can make a control system more scalable and modular, their cost is significant compared to the motherboard. Currently, passive backplanes are only used in higher end machining applications.

Operating System

GM Powertrain desired an operating system that could support real time applications such as motion control while also supporting the Microsoft Windows^{TM} environment. Microsoft Windows^{TM} is the dominant operating system in use on personal computers. Numerous applications are being written to run on the Windows 95/NT^{TM} operating systems. Microsoft Windows^{TM} would be ideal for machine tool control except it lacks real time capability. The hopeful future is for Microsoft to develop real time functionality to the either the Windows 95^{TM} or Windows NT^{TM} operating systems. Currently, QNX^{TM} is a commercially available PC based real time operating system. QNX^{TM} can support both text and graphically based interfaces for an operator. The unfortunate aspect to using QNX^{TM} is lack of software support for various hardware. The availability of software drivers to support various pieces of hardware is less due to small demand compared to the demand for Microsoft Windows' drivers.

Graphical User Interface

Operators of modern CNC machine tools use a graphical user interface (GUI) instead of a hardwired keypad. Although GUI's are a standard item on a CNC machine tool, how they are organized and used is not. Significant resources are used in training operators to use the GUI's on various machine tools. Therefore, standardizing GUI's would eliminate redundant training which would increase productivity. GUI's should be designed to be intuitive in

nature and should be complete with help icons and pull down menus. Additionally, GUI's should embrace the de facto standard of the day. For example, most computer users today are familiar with the Windows 95/NT graphical user interface. Designing a GUI around Windows 95/NT^{TM} would immediately increase user comfort with the interface.

Network Connection

GM Powertrain desired to have each machine tool be able to communicate with other machine tools and manufacturing controllers in order to increase manufacturing efficiency. Many manufacturing lines are operated using programmable logic controllers (PLC) in order to coordinate functions between processes. It would be ideal for each machine tool to communicate with these PLCs and other machine tools over a standard network connection such as the Ethernet. Having a network connection could also aid in communication between the engineering design department and the manufacturing floor. Design changes could be made in engineering design and the new part programs downloaded to the machine tools.

In conclusion, GM Powertrain seeks to have standardization so that different hardware components can be interchanged. This will allow machine tool capabilities to be scaled and customized at a reasonable cost. At the same time, GM Powertrain does not wish to have OMAC to become too rigid thereby discouraging the implementation of new technology when it develops.

2.3.2 MDSI

In 1989 Manufacturing Data Systems Inc. (MDSI) was founded by Charles Hutchins and Bruce Nourse. Their goal was to have an open exchange of information between the PC networks of a company and the machine tools on the factory floor. They also endeavored to give companies a choice from simply accepting the proprietary controls provided by a machine tool manufacturer. In 1993 OpenCNC^{TM} was introduced as an open architecture control system.

OpenCNC^{TM} is a PC based control system for machine tools. All the control algorithms are performed using a standard Intel processor. Since OpenCNC^{TM} was written to perform

all the real time calculations inside standard PC hardware, this eliminates the proprietary hardware made by machine tool manufacturers including specialized PC motion cards. Furthermore, OpenCNC^{TM} allows a company to put the same controls on different brands of machine tools and standardize the interface between the engineering department and the factory floor. OpenCNC^{TM} can provide similar operator interfaces so that machine operators would be more comfortable running machine tools made by different companies. The strength of OpenCNC^{TM} is using standard computer hardware mixed with the flexibility of software based control defined by the end user [7].

OpenCNC^{TM} has also made retrofitting machines a lucrative choice for manufacturers. Great Lakes Industries' (GLI) Vice-President and General Manager Don Werner explains, "OpenCNC enabled us to go into the used-equipment market and buy machines with obsolete controls, retrofit them, and make them operational at a fraction of the cost of a new machine." [8] For an example Werner states, "We've looked at gear hobs for years, but new machines can cost $1 million. Last year we bought a gear hobber that wasn't working because of a bad control for $50,000. After conversion to OpenCNC, we've been in production ever since." [8] GLI has refitted 15 machine tools and can perform retrofits themselves in between 48 to 72 hours. [8]

2.4 Future Speculation

Open architecture control appears to be the future of machine tool control. PC hardware is steadily becoming faster and more robust. Since the market for PC hardware is tremendous, specialty control items such as PLCs and motion control cards will find it more difficult to compete against the PC. Therefore, machine tool builders will begin to market their products according to the options available in their control software. Machine tool builders may begin producing their own PC based open architecture control software or they could establish relationships with machine tool software producers. Open architecture software packages will be evaluated according to how easily their software can be networked with other computers, machine tools, and corporate data bases.

Chapter 3

Milling Machine Control

A typical 3 axis CNC milling machine, as shown in Figure 3.1, is capable of moving in 3 directions: X, Y, and Z. A piece of stock is placed on the machine tool table to be machined to a desired shape. The X and Y axes or slides are capable of moving the machine tool table in the horizontal plane. The Z axis moves the tool to make contact with the stock in order to make machining cuts. The A axis turns the spindle and tooling piece. The spindle refers to the hardware necessary to spin the tool around the A axis. The A axis rotates around the Z axis. An A+ movement refers to a counter clockwise movement of the spindle while A- refers to a clockwise movement of the spindle.

In order to fully define how a milling machine operates, the CNC mill will be broken into several parts as shown in Figure 3.2. The first operation performed on a CNC machine tool at startup is homing or syncing the machine tool. The homing operation consists of manually moving the axes to predetermined point called the home position. All movements of the machine tool will be tracked relative to this fixed point. From this point the CNC control can operate in both manual and automatic modes. In manual mode the operator manually commands the milling machine to move the axes and turn the spindle. In automatic mode the CNC mill reads a set of sequential functions from a part program. Part programs consist of G and M codes. G codes refer to motion commands and cause the coordinated movement of the axes to perform a specific cut on the work piece or stock. M codes perform nonmotion commands for miscellaneous functions such as setting the spindle and turning the coolant on and off.

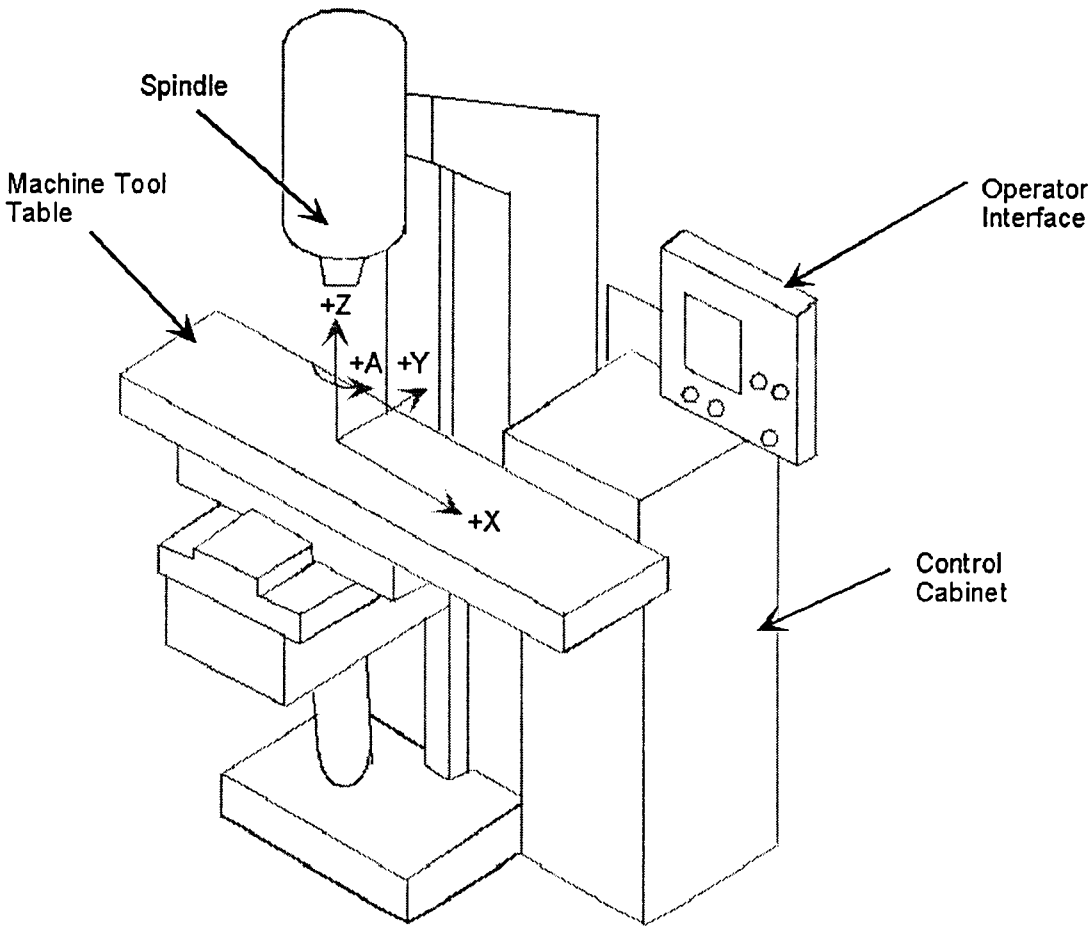


Figure 3.1: Coordinates of a 3 Axis Milling Machine

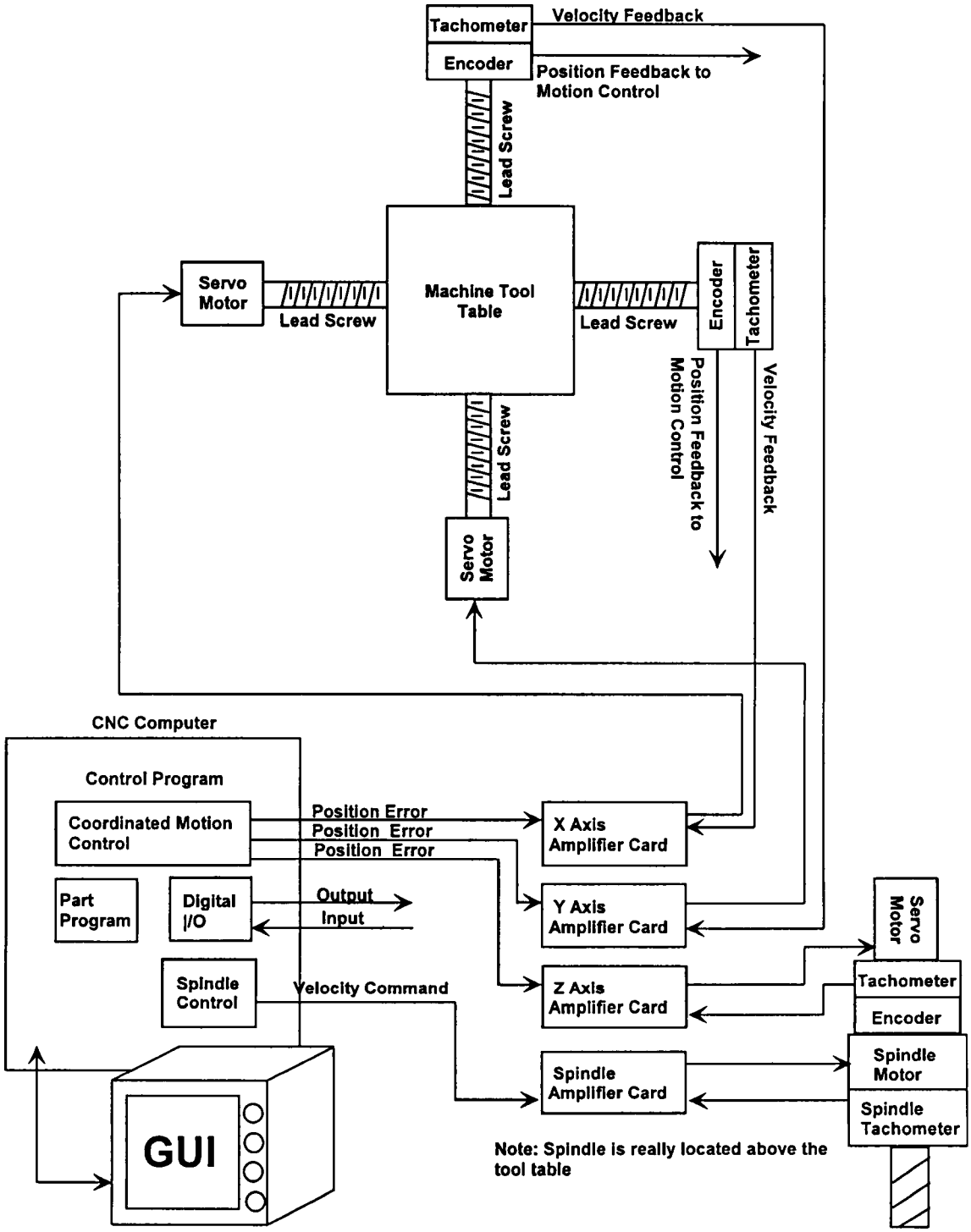


Figure 3.2: CNC Control for a 3 Axis Milling Machine

After the computer receives its instructions by manual or automatic methods, the computer must send signals to various hardware on the machine tool. For motion control the computer calculates a trajectory to move the axes which is referred to as path planning. The computer then sends a position commands or command voltage to each of the axis amplifier cards. Each axis amplifier performs velocity feedback control for its axis using tachometers. After comparing command voltage to the velocity feedback signal, the amplifier card adjusts the voltage to the DC servo motor. The DC servo motors turn the lead screws, thereby moving the part relative to the tool. Position feedback is performed within the computer. An encoder is attached to the lead screws in order to determine the position of the part relative to the tool.

For the spindle the computer calculates a desired spindle speed and sends a velocity command to the spindle amplifier card. The spindle amplifier performs velocity feedback control for the spindle. Typically the velocity feedback is from a tachometer.

Digital inputs and outputs (DIO) must be observed for the machine tool to function properly. DIO consists of devices with two states: on and off. Examples of digital inputs are home switches, overtravel switches, level switches, and pneumatic switches. Examples of digital outputs are relays and solenoids.

3.1 Single Axis Motion Control

The motion control of the machine tool table and the cutting tool is crucial for quality manufacturing. This section will discuss the control of one axis only while Section 3.2 covers coordinated axis motion control. The coordinated axis motion control produces a desired position for each of the axes $P(s)$ as shown in Figure 3.3. The position is compared to the actual position of the axis determined from the encoder position feedback and sends a voltage command to the axis amplifier card. The axis amplifier card scales the position command voltage before sending a voltage to the DC servo motors. Axis amplifier cards support velocity feedback to increase system stability. The DC servo motors drive the lead screws to the desired position $d(s)$.

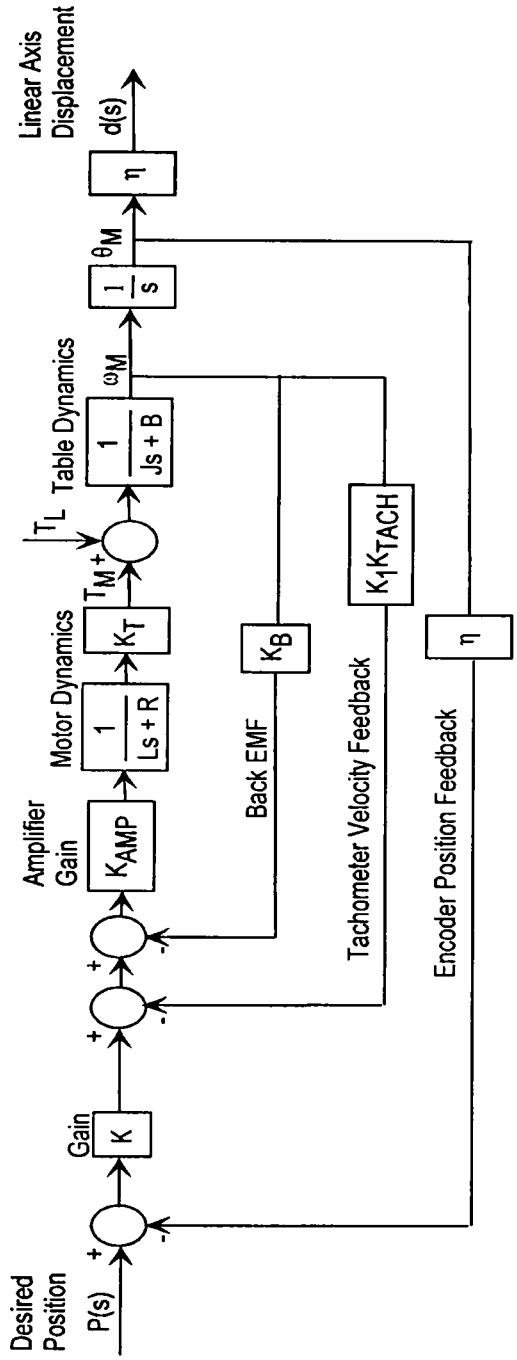


Figure 3.3: Single Axis Control Block Diagram

Subsection 3.1.1 will discuss the operation of optical encoders for position feedback. Subsection 3.1.2 reviews the functionality of the amplifier card and the velocity feedback to the amplifier card using a tachometer. Subsection 3.1.3 will discuss the transmission between the DC servo motor and the linear movement of the axes. Subsection 3.1.4 discusses the use of DC Servo motors to provide axis motion. Subsection 3.1.5 models of the overall axis drive system and an overall block diagram for the a single axis drive system as shown in Figure 3.3.

3.1.1 Encoders

Encoders are sensors used to determine the angular position of a shaft. Encoders can be capacitive, magnetic, contact, or optical in nature. However, most encoders used on modern machine tools are optical. An optical encoder shines a light through a disc with slits around the perimeter. The disc rotates with the shaft, and a light sensor monitors the light passing through the disc. Electronic circuitry in the encoder reads the number of slits that pass by the light source, thereby determining the angular movement.

Optical encoders are of two types: absolute and incremental. An absolute encoder has a unique digital output for each angular position. Therefore, to have high displacement resolution, a significant amount of hardware is necessary. Although these encoders are reliable, the hardware costs limit their application on machine tools.

Typically, machine tools use optical incremental quadrature type encoders. An incremental quadrature encoder produces a square wave output. Figure 3.4 shows channels A, B, and I. When channels A and B go high, the sides of the square wave are used to produce a pulse output. Each pulse marks an incremental angular displacement. The Index or I channel is a reference or marker pulse. The marker pulse signals one complete revolution by the encoder disc. The position of phase A with respect to phase B indicates the direction the encoder is turning. If phase A leads phase B, the encoder is counting forward. Likewise, if phase A lags phase B, the encoder is counting in the reverse direction. Quadrature encoders are specified in terms of lines of resolution. Equation 3.1 relates the number of lines to the number of pulses produced for a quadrature type encoder. The number of pulses determines

the resolution of the encoder. If an quadrature type encoder had 1024 lines of resolution, the angular position of the encoder could be determined to $\frac{2\pi}{4096} = 0.0015$ radians.

$$pulses = 4 * lines \tag{3.1}$$

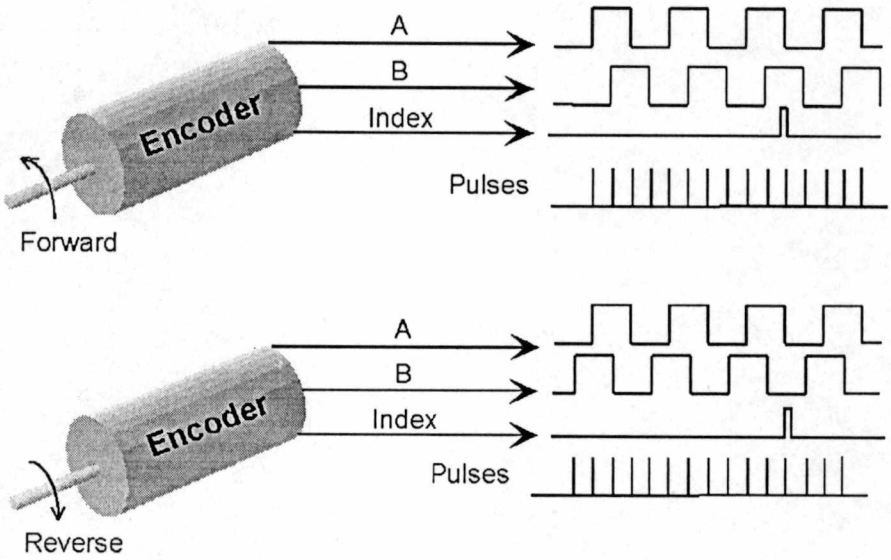


Figure 3.4: Encoder Direction Determination

Although the incremental encoder allows for an inexpensive, high resolution angular measurement, they do have some technical problems. In the event of a power failure, the position of the table may move without the encoder counting the displacement. This event will force the machine to be synced again because the encoder may have moved without the computer being aware of the displacement. Optical encoders also require the circuitry to count the number of pulses during actuation. If the DC motor puts out a high velocity, the circuitry must be designed to read the pulses quickly enough to prevent unknown motion of the axis.

3.1.2 Amplifier Cards and Tachometers

Amplifier cards are used to increase the position command coming from the computer and to output a voltage to drive the DC servo motors. Most amplifiers have velocity feedback

capability to increase the system damping. The amplifier cards typically have three control adjustments: K_{AMP}, K_1, and the balance potentiometer which can be observed in Figure 3.3. K_{AMP} is a proportional gain to increase the difference between the command voltage and the tachometer feedback voltage. Increasing K_{AMP} will increase the system's rise time. K_1 is the velocity feedback gain and is used to increase the size of the of the voltage coming from the tachometer providing velocity feedback. Increasing K_1 will add damping to the system. The balance potentiometer is used to null out any offset voltage from the computer.

Tachometers are sensors used to determine shaft speed and to provide velocity feedback to the system. Tachometers are specifically designed and specified to ensure the angular velocity is proportional to the output voltage for a certain speed range as shown in Equation 3.2.

$$V_{TACH} = K_{TACH}\omega_{TACH} \tag{3.2}$$

Where,

K_{AMP} = proportional gain

K_1 = amplifier gain in the velocity feedback path

V_{TACH} = tachometer output voltage

K_{TACH} = tachometer constant

ω_{TACH} = angular velocity of the tachometer

3.1.3 Transmission

Although DC servo motors can supply motion in a controllable form, they sometimes lack the torque capability necessary to move the required loads. At the same time DC servo motors supply angular motion while the tables need to be actuated in a linear fashion. In light of these two needs, many machine tools utilize a mechanical speed reduction to provide the needed torque and linear motion.

Various transmissions have been used to actuate the slides of the table. Typically a DC servo motor turns either a gear or belt driven speed reduction. The output of the speed

reduction turns the lead screw which provides linear actuation of the axis. Figure 3.5 shows a typical transmission with a gear driven speed reducer driving a lead screw. Equation 3.3 shows the overall relationship between the DC motor angular displacement, θ_M , and the linear displacement of the table, d.

$$\eta = \frac{d}{\theta_M} \tag{3.3}$$

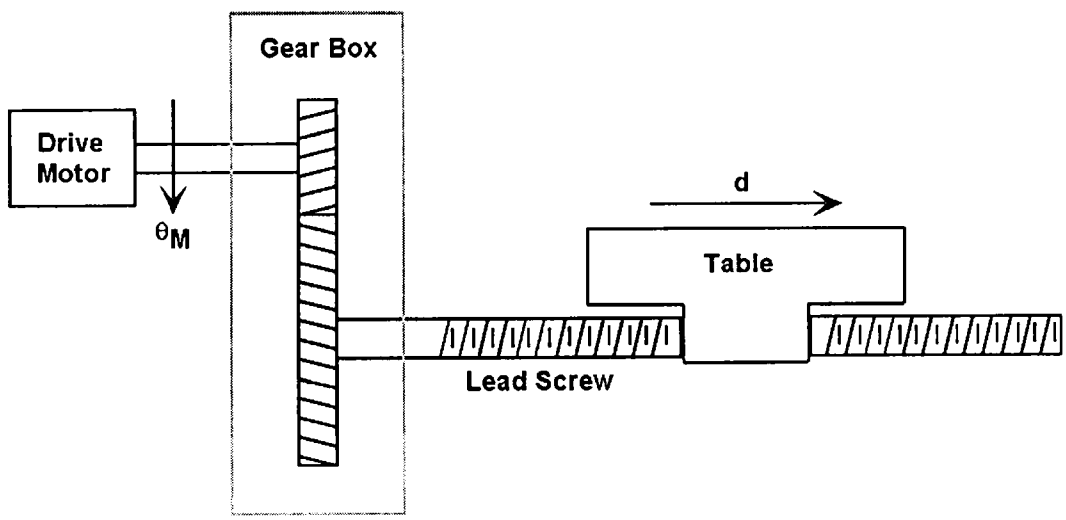


Figure 3.5: Machine Tool Transmission

3.1.4 DC Servo Motors

DC servo motors are the most prevalent way to actuate the slides of a milling machine [9] [10]. A DC servo motor generally refers to permanent magnet, armature winding, brushed DC motor. The permanent magnets create a constant magnetic field across the armature windings. The current, I, changes directions while traveling through the armature windings. The current induces an electromotive force (EMF) or torque on the shaft of the motor. The torque for a DC servo motor can be expressed in Equation 3.4.

$$T_M = K_e \Phi I sin(\theta) \tag{3.4}$$

Where,

T_M = motor torque

K_e = motor constant

Φ = magnetic flux density

I = armature current

θ = angle between the magnetic field and armature current vectors

The DC servo motor is designed so the angle between the magnetic field from the permanent magnet and the current vectors is 90^o. Therefore, the motor torque, T_M, is linearly proportional to the current, I, by the torque constant, K_T.

$$T_M = K_T I \tag{3.5}$$

Where,

$$K_T = K_e \Phi sin(90^o) = K_e \Phi \tag{3.6}$$

Applying Kirchoff's Voltage Law to the armature circuit shown in Figure 3.6:

$$V - RI - L\frac{dI}{dt} - K_b \omega_M = 0 \tag{3.7}$$

Where,

V = motor voltage

R = armature resistance

L = armature inductance

K_b = back emf constant

$\omega_M = \dot{\theta_M}$ = angular velocity of the motor

Converting Equation 3.7 to the Laplace domain

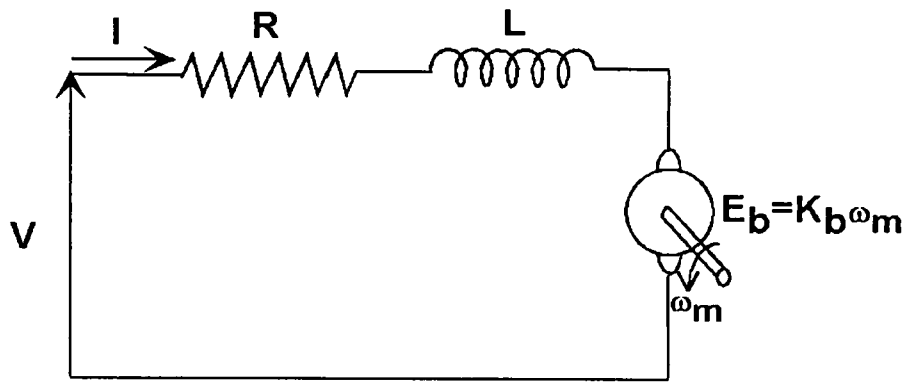


Figure 3.6: DC Armature Circuit

$$\frac{I}{V - K_b s\theta_M} = \frac{1}{Ls + R} \tag{3.8}$$

3.1.5 System Modeling

Applying Newton's Second Law to the DC motor with the transmission and load torques as shown in Figure 3.7, the transfer function for an individual axis can be determined.

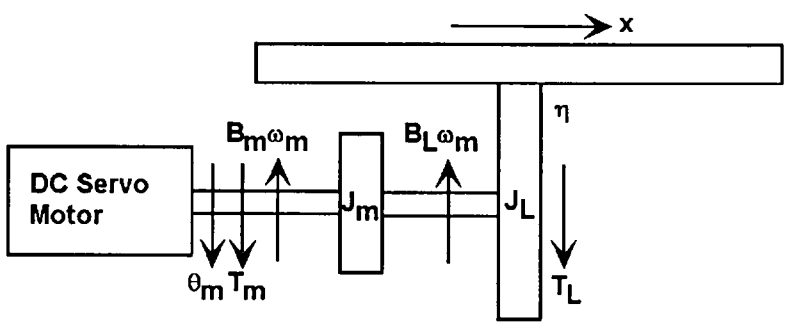


Figure 3.7: Free Body Diagram of Single Axis

$$J_M\ddot{\theta_M} + J_L\ddot{\theta_M} = T_M - B_M\dot{\theta_M} - B_L\dot{\theta_M} + T_L \tag{3.9}$$

Where,

J_M = motor inertia

J_L = load inertia due to the gear reduction

B_M = viscous friction coefficient of the motor

B_L = viscous friction coefficient of the gear reducer

T_L = the externally applied load torque

T_L can be caused by cutting forces between the tool and the part as well as unmodeled friction forces in the motor and transmission systems. Let,

$$J = J_M + J_L \tag{3.10}$$

$$B = B_M + B_L \tag{3.11}$$

Combining 3.9, 3.10, and 3.11 while converting to the Laplace domain

$$\frac{\omega_M}{T_M + T_L} = \frac{1}{Js + B} \tag{3.12}$$

The amplifier card is inserted to increase the command voltage from the computer. The tachometers can be inserted to provide velocity feedback for the system. Using the encoders with the mill computer, position feedback can be inserted with a proportional gain K. K is used to scale the following error where the following error is defined as difference between the desired position, $P(s)$, and the actual position $d(s)$. Combining the Equations for the DC servo motor, amplifier cards, and the encoder position feedback, the control block diagram for a single axis system can be determined as shown in Figure 3.3.

With the configuration in Figure 3.3 there are three control variables that can be adjusted to influence system performance: K, K_{AMP}, and K_1. K and K_{AMP} will increase the system's rise time. K_1 tends to add damping to the system. CNC machine tools are tuned to be overdamped and to minimize the position or following error. Overshoot would result in cutting away too much material while a large following error would result in a significant undesired offset between the desired position, $P(s)$, and the actual position, $d(s)$.

3.2 Coordinated Motion Control

Two strategies are used to coordinate the motion between the three axes depending on the machining operation to be performed. Point to point and continuous path or interpolated motion are used to perform a discrete series of tooling operations. Suppose a series of holes need to be drilled into a plate: point to point machining could be used to repeat the same drilling operation in several places. Interpolation is used to machine various shapes and contours. Various interpolation methods are used depending on the difficulty in representing the tooling trajectory. Trajectories with mathematically defined paths can be produced with less code than arbitrarily defined tool paths.

3.2.1 Point To Point Machining

Point to point (PPT) machining is a discrete method for performing operations such as drilling, boring, reaming, and tapping in which processing tasks are executed only at motion end points [9]. The path taken between the end points has no bearing on the process. Suppose a simple plate in Figure 3.8 needed to have five holes, "a" through "e", drilled. The same procedure would need to be followed to drill each hole into the plate. The following outlines how PPT operations and specifically the plate drilling operations could be conducted:

1. Positioning. The tool is positioned above the hole center without any machining taking place. (The drill bit is positioned above hole a using X and Y axis control only.)

2. Machining. The machining operation is brought into contact with the workpiece, usually along the Z axis, at a specified spindle speed, feed rate, and hole depth. (The drill bit drills hole a.)

3. Retracting. The tool is retracted from the workpiece once the desired hole depth has been reached. (The drill bit is removed from hole a.)

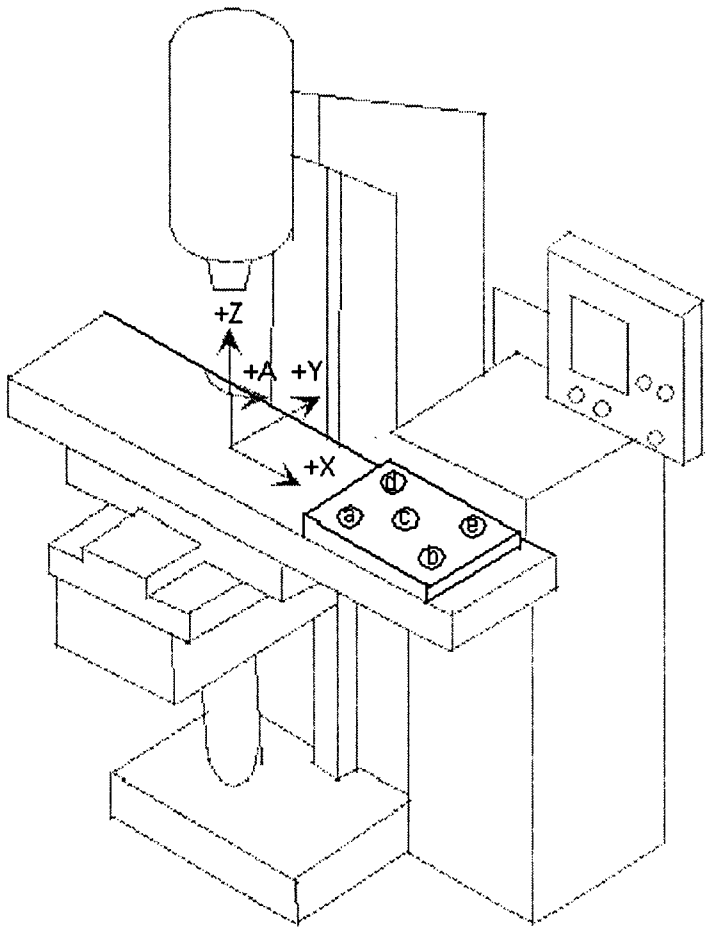


Figure 3.8: Point to Point Machining

4. Repeat. The tool is repositioned above the next hole center to repeat the operation. (The procedure is repeated to drill holes b through b.)

Since positioning and machining are done in two independent steps, only two axes are ever controlled at once. This allows for simple programming while performing PPT type operations. The PPT cannot be used for cutting arbitrarily shaped surfaces.

3.2.2 Continuous Path Machining

Continuous path machining allows a part to be shaped or contoured. The part is shaped by the tool traversing a series of straight line segments or arcs. Since these line or arc segments can be arbitrarily defined in 2D or 3D space, the control system must be able to control multiple axes simultaneously while the tool is in contact with part. Most milling machines are capable of three axis contour machining where an arbitrary line or arc can be traversed in 3D space.

Interpolation is the process of breaking a desired tool path into a series of small lines and arcs. Although interpolation will not produce the exact desired path, it can provide an acceptable approximation. In general, the more discrete lines or arcs used to define a path, the closer the interpolated path is to the desired tool path.

Linear Interpolation

Linear interpolation, also known as spatial interpolation, is the process of traveling a specified tool path by approximating the path by a series of short straight lines. Linear interpolation can be done for 2D and 3D tool trajectories. Suppose a part needs to be cut along a straight line in 3D space from the starting point S (1,2,3) and ending at point E (4,4,9) as shown in Figure 3.9. The straight line path is produced by determining the incremental distances along each axis. The straight line will be produced if each axis takes the same amount of time to go from S to E. The incremental distances can be calculated by

$$\delta x = 4 - 1 = 3 \tag{3.13}$$

$$\delta y = 4 - 2 = 2 \tag{3.14}$$

$$\delta z = 9 - 3 = 6 \tag{3.15}$$

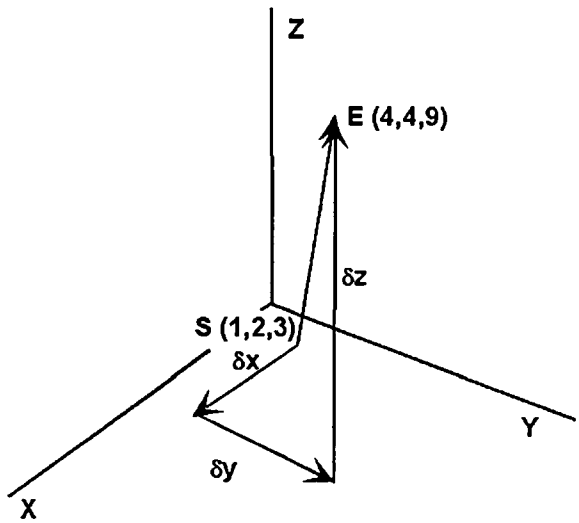


Figure 3.9: Linear Interpolation

Each of the individual axes is moved by a series of pulses. Typically milling machines move 0.0001 to 0.0002 inches per pulse. Equations 3.14, 3.15, and 3.15 define the ratio of the number of pulses between the axes. There must be 1 pulse in Y for every 3 pulses in Z and there must be 1 pulse in X for every 2 pulses in Z. The pulses must be given simultaneously to traverse a line in 3D space. Linear interpolation is used to approximate straight lines and arbitrary shapes that cannot be defined by a mathematical equations.

Circular Interpolation

Circular interpolation is a method of approximating an arc by a series of short straight lines. An arc could be approximated using a series of linear interpolations producing a cut with the same accuracy as a circular interpolation. However, the circular interpolation uses the knowledge that the path is an arc to reduce the number of program blocks necessary for the cut. For example a linear interpolation of an arc could require an order of 100 more program blocks to produce the same cut as circular interpolation.

3.3 Spindle Drives

Typical spindle drives come with an amplifier card. The computer sends a velocity command to the spindle amplifier card. The amplifier card increase the velocity command and performs velocity feedback control for the spindle. The voltage coming from the computer to the amplifier card would be adjusted to change the spindle speed. A tachometer is typically used for velocity feedback. Figure 3.10 shows the block diagram for spindle speed control using an amplifier card.

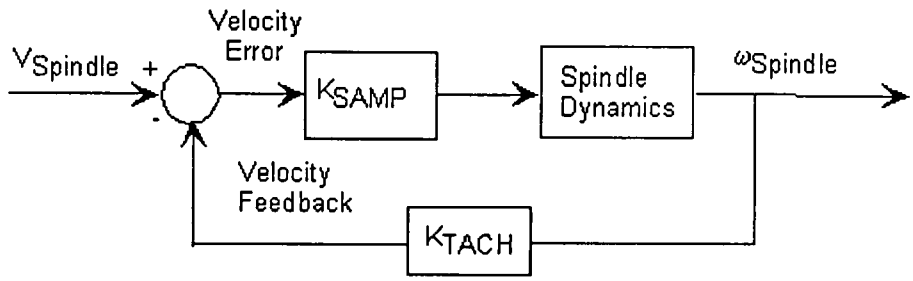


Figure 3.10: Spindle Control Block Diagram

Where the variables in the block diagram are:

$V_{Spindle}$ = spindle velocity command from the computer

K_{SAMP} = spindle amplifier gain

K_{TACH} = tachometer constant

$\omega_{Spindle}$ = spindle velocity

3.4 Digital Inputs and Outputs

Digital inputs and outputs control and monitor hardware with two states: on and off. Examples of digital inputs are limit switches, level switches, and pneumatic switches. Examples of digital outputs are relays, switches, and solenoids.

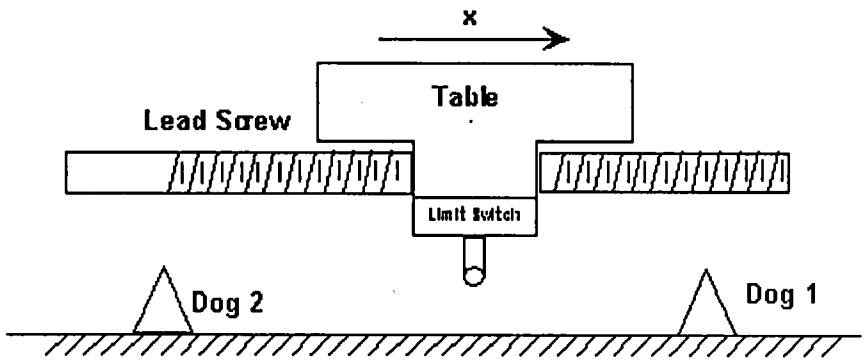


Figure 3.11: Overtravel Switch Configurations

3.4.1 Digital Inputs

Overtravel Switches

Limit overtravel switches are used to determine if the table has been driven too far along the lead screw. Overtravel switches are a hardware safeguard to prevent damage to the lead screws and the axes. Figure 3.11 shows a single limit switch attached to the moving table. The two dogs, used to open the overtravel switch when the table has traveled too far, are stationary. With only one overtravel switch, the logic cannot determine which direction (+/-) the table has overtraveled.

Home Switches

Home switches are used determine when the machine tool is in the home position. The home switches define an inertial reference frame from which movements can measured. Furthermore, these switches allow the machine tool to have the same starting point each time the machine tool is turned on. Each axis has its own home position limit switch. Typically, an operator will manually move the axes to its respective home switch. Next, the control will move the slides in order to determine where the encoder's marker pulse is with respect to the home switch. This will define the home position in terms of encoder pulses.

Level and Pneumatic Switches

Level switches are used to determine whether fluids such as waylube and coolants are in sufficient amounts for normal machine tool operation. Pneumatic switches determine if sufficient air pressure is present to operate the pneumatic devices on the machine tool.

3.4.2 Digital Outputs

Digital outputs control devices such as relays and solenoids. When a relay receives a voltage to its magnetic coil, the magnetic field either opens or closes a set of contacts on the machine tool. Relays allow the control to turn various electrical hardware on or off. When a solenoid has a voltage applied to its magnetic coil, the magnetic field moves a spool to allow a fluid or gas to actuate some device. A brake solenoid is used to apply a brake to a spindle to stop its motion. Solenoids allow the control to turn various mechanical systems on or off.

Chapter 4

Conversion to Open Architecture Control

Chapter 4 covers the conversion of a Bridgeport Series 1 V2XT milling machine to open architecture control. MDSI's OpenCNC^{TM} was chosen as the commercially available open architecture system to be installed on the Bridgeport mill in the University of Tennessee's Manufacturing Systems Laboratory.

Figure 4.1 shows the OpenCNC^{TM} Control Configuration. This configuration uses two computers, one acting as a server and the other as a client. The server is used to develop the soft logic programs for the milling machine, to receive part programs from other computers on the network, and to download the operating system and necessary files to run the mill. The server contains two operating systems: Windows 95^{TM} and QNX^{TM} . QNX^{TM} is a real time operating system for PC hardware. The Windows 95^{TM} operating system uses a soft logic development tool called ISaGRAF^{TM} [11]. Soft logic refers to logic developed in software versus hardware. The soft logic was custom developed to control the Bridgeport milling machine. Once the soft logic is developed, it is transferred to run under the QNX^{TM} operating system.

Although the soft logic is developed on the server, the client has real time control of the mill. If the client is on the network, the server down loads the QNX^{TM} operating system, MDSI's OpenCNC^{TM} and the soft logic to the client computer. After down loading, QNX^{TM} will run the custom soft logic developed in ISaGRAF^{TM} with the OpenCNC^{TM}

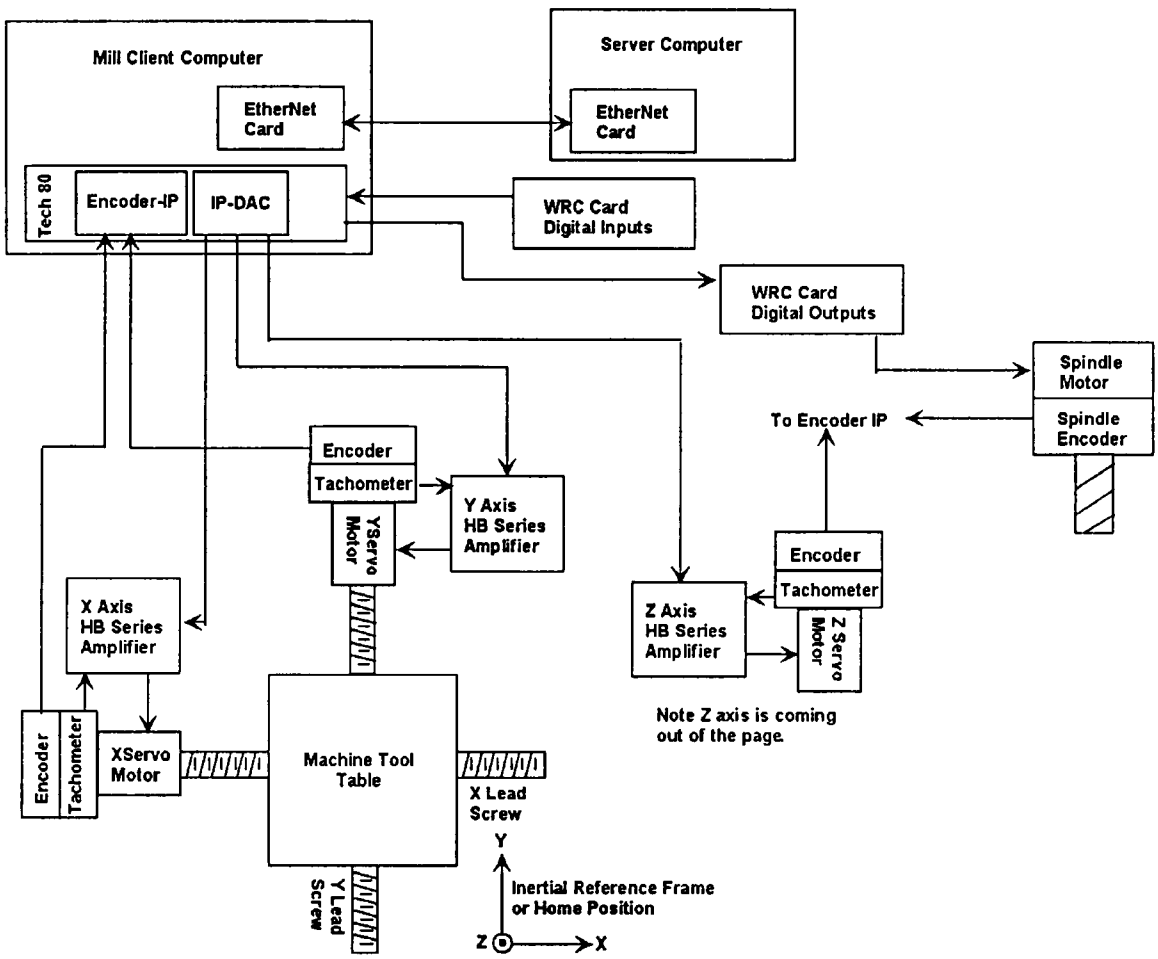


Figure 4.1: OpenCNC Control Configuration

control program. The client can boot from the server or it can boot up using its own hard disk. The client contains a Technology 80 carrier card (Tech 80) [12] used to interface between the client PC and the milling machine hardware. Western Reserve Controls [13] cards are used to monitor the digital inputs and outputs. The Western Reserve Control cards interface with the Tech 80 card over a standard 60 pin ribbon cable. The Tech 80 contains two industry packs: Industry Pack Digital to Analog Converter (IP-DAC) and the Encoder Industry Pack (Encoder-IP). The IP-DAC converts the digital command signal from the client computer to an analog signal that is sent to the HB Series Amplifier. The HB Series amplifier increases the command signal from the computer and performs velocity closed loop control for its respective axis. The position feedback is performed within the OpenCNC^{TM} software. The Encoder-IP reads the encoders to determine the position of the axes in 3D space thereby providing position feedback for OpenCNC^{TM} .

4.1 Chapter Format

The text of this chapter was formatted to aid in readability. The digital input and output sections contains hardware names and software names. For example, X_REF is the hardware name for the X axis home or reference switch and is named with only capital letters. **DI_X_REF** is the software variable that monitors the state of the X_REF switch. Digital input and output software variables will labeled with bold capital letters in block font. OpenCNC^{TM} contains several software variables that can modified. For example **axFErrGain** adjusts the following error gain. OpenCNC^{TM} variables will be in bold block font. This chapter contains several files used by OpenCNC^{TM} such as **tunebgp1**. these files and their paths will be typed with a bold font.

4.2 Axis Definitions

The Bridgeport Series 1 V2XT milling machine is a 3 axis CNC machine tool [14]. Each of the axes are given an axis number used by OpenCNC^{TM} as shown in Table 4.1. These

numbers will be used in the **tunebgp1** file discussed in Section 4.6. Figure 4.2 shows the machine tool table and the axis definitions for the Bridgeport mill. The X and Y axes are associated with the machine tool table movement in the horizontal plane. The Z axis is associated with the vertical position of the tool relative the machine tool table. The A axis defines the direction of tool rotation. A+ corresponds to a counter clockwise rotation of the spindle while A- corresponds to a clockwise rotation of the spindle.

Table 4.1: Axis Numbers

Axis	OpenCNC Axis #
X	0
Y	1
Z	2
A	3

Each axis has a 2:1 speed reduction from the motor to the lead screw. Each lead screw has a pitch of 0.2 inches per revolution [14].

4.3 Axis Motion Control

Figure 4.3 shows a control block diagram for a typical translational control. The spindle control will be discussed in Section 4.4. OpenCNC^{TM} commands a desired position for an axis and compares it to the actual position of the axis. The difference, the position error, is scaled by three variables in OpenCNC^{TM} . The **axFErrGain** variable is a linear gain used to scale the position error between the desired and actual positions. The **defAccelMax** and **defJerkMax** are used to limit the control action from producing large accelerations and jerks in any axis. Large accelerations and jerks can result in significant forces between the tool and the part during manufacturing. This could result in part damage or tool destruction and should be avoided. **axFErGain**, **defAccelMax**, and **defJerkMax** are all defined in the tunebgp1 file discussed in Section4.6. OpenCNC^{TM} sends an analog voltage, V_{IP-DAC}, or position command from the computer to the HB amplifier cards. The HB amplifier

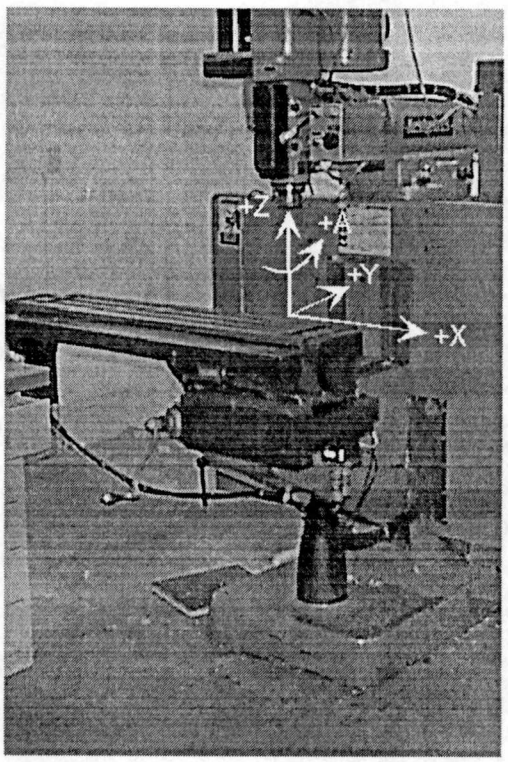


Figure 4.2: Coordinate Definition

cards are the original amplifier cards on the Bridgeport mill; they provide velocity feedback control for each axis. The HB amplifier receives an actual velocity signal from the axis tachometers and scales the difference by K_{AMP} to produce, V_{MOTOR}, which is the actual voltage sent to the servo motors. The actual position of an axis is measured by a 250 line optical incremental quadrature type encoder [14]. The position feedback is performed within the OpenCNC^{TM} software.

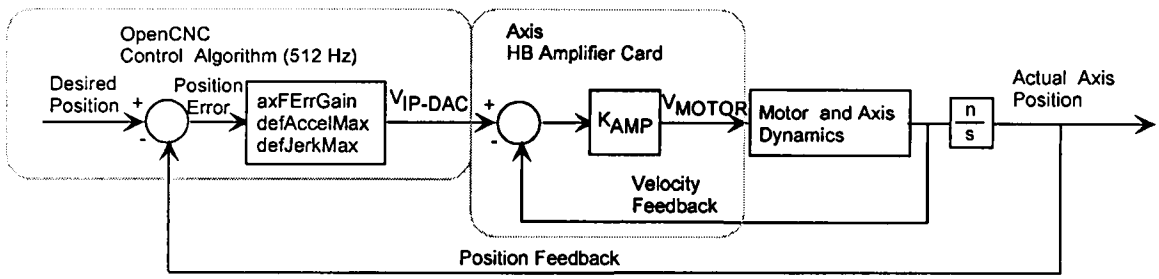


Figure 4.3: Axis Control Block Diagram

All of the axis control loops are closed at a rate of 512 Hz. With the 2:1 speed reduction and a lead screw pitch of 0.2000 inches per revolution, the position of each axis can be determined to within 0.0001 inches. The axes were tuned to have less than 0.1 inches of position error at a feed rate of 100 inches per minute. Tuning procedures are discussed in [15].

4.4 Spindle Control

The spindle on the Bridgeport milling machine is actuated by a Emerson Electric 208 VAC three phase motor [14]. The spindle has two gear ranges: low and high. **DI_LOWGEAR** is a 24 VDC input, further discussed in Subsection 4.5.1, that monitors a switch called LOWGEAR to determine which gear range has been selected. **DI_LOWGEAR** is a boolean variable with two states: on or off (True or False). If **DI_LOWGEAR** is true, then the spindle is in low gear. If **DI_LOWGEAR** is false, then the spindle is in high gear. The spindle speed is adjusted by a variable pitch pulley.

Two solenoids, SSINC and SSDEC, control the speed ratio between the motor and the

spindle. DO_SSDEC and DO_SSINC are 24 VDC outputs to control the solenoids SSDEC and SSINC respectively and are further discussed in Subsection 4.5.1. The SSDEC decreases the spindle speed while the SSINC increases spindle speed.

The spindle turn direction is controlled by two sets of contacts SPINDLE_CW and SPINDLE_CCW. SPINDLE_CW and SPINDLE_CCW, like LOWGEAR, are controlled by two digital outputs called **DO_SPINDLE_CW** and **DO_SPINDLE_CCW** discussed in detail in Section 4.5.1. When the spindle is in low gear with the SPINDLE_CW contacts closed, the spindle turns in the clockwise direction. A low gear setting with the SPINDLE_CCW contacts closed causes the spindle to turn in the counter clockwise direction. When the spindle is high gear, the situation is reversed. When the spindle is in high gear with the SPINDLE_CW contacts closed, the spindle turns in the counter clockwise direction. A high gear setting with the SPINDLE_CCW contacts closed causes the spindle to turn in the clockwise direction. The spindle has a solenoid brake controlled by a 24 VDC output called **DO_NOT_BRAKE**. If **DO_NOT_BRAKE** is true, the brake is not applied.

Figure 4.4 shows the spindle control strategy. The desired spindle speed is evaluated to determine if the desired spindle speed is valid for the gear range selected by LOWGEAR. In low gear the spindle speed ranges from 60 to 480 revolutions per minute (RPM). In high gear the spindle speed ranges from 480 to 4280 RPM. If the spindle speed request is not valid, the spindle will stop until a valid spindle speed request is given. If the spindle speed request is valid, the program will set **DO_SPINDLE_CW** and **DO_SPINDLE_CCW** appropriately to turn the spindle in the correct direction. If the actual spindle speed is not within $\pm 10\%$ of the desired spindle speed, SSINC and SSDEC will be adjusted to bring the actual spindle speed within $\pm 10\%$ of the desired spindle speed. The spindle logic is controlled at a rate of 64 Hz.

4.5 Hardware Configuration

Although OpenCNC^{TM} does not use proprietary hardware to control the mill, it requires a standard set of industrial hardware. The Bridgeport V2XT milling machine conversion used

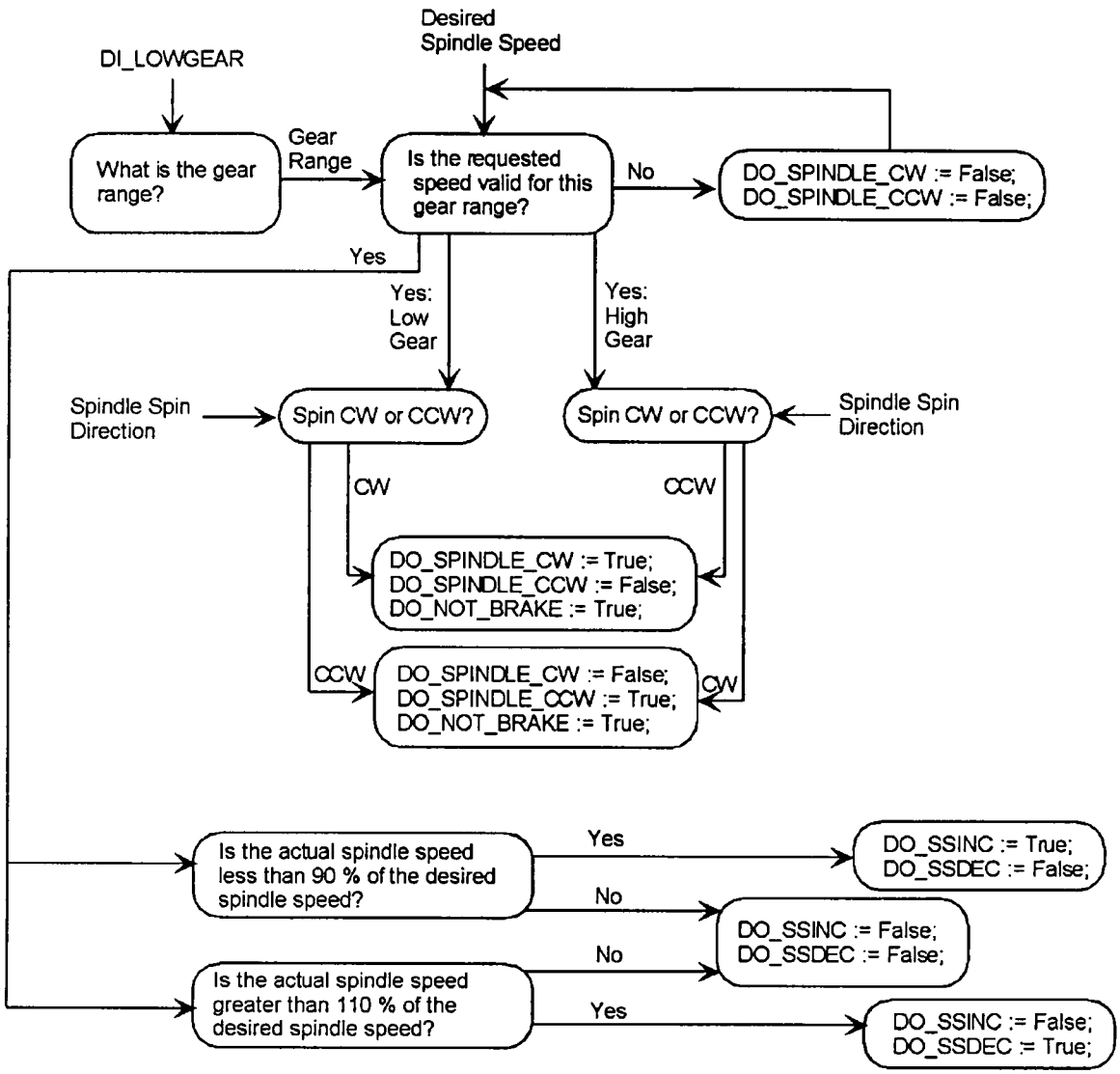


Figure 4.4: Spindle Control Block Diagram

the Tech 80 Industry Pack configuration discussed in [13]. The Tech 80 Model 5641 card connects with the client PC next to the machine tool using a standard ISA connection as shown in Figure 4.5. The Tech 80 card provides the means for the PC to control the various inputs and outputs on the mill. OpenCNC^{TM} monitors and controls three different types of inputs and outputs on the machine tool: Digital Inputs and Outputs, Analog Outputs, and Encoder Inputs. Figure 4.6 shows the location of the hardware in the Bridgeport cabinetry.

Digital inputs and outputs have two states: on and off. The digital inputs and outputs (DIO) consist of limit switches, pressure switches, and relays to monitor and control various discrete functions on the mill. The Western Reserve Control cards are used to control the DIO on the machine tool and communicates with Tech 80 card via a 50 pin ribbon cable to the breakout board.

The analog outputs control the axis drive system. The client PC calculates the position for each axis and sends out a digital position command. The Industry Pack Digital to Analog Converter (IP-DAC) converts the digital command signal to an analog signal and sends it to the HB Series Amplifiers. The HB Series Amplifier cards amplify the position command to the servo motors and perform velocity feedback control for the servo motors. The IP-DAC was inserted on to the Tech 80 carrier card. The HB Series Amplifiers were wired into the IP-DAC terminal block inside the control cabinet. The IP-DAC and the HB Series Amplifier communicate over a standard 50 pin ribbon cable. Subsection 4.5.4 covers wiring the IP-DAC as well as the operation of the HB Series Amplifier.

An encoder determines the angular position and velocity of the drive screws which move the machine axes. The Encoder Industry Pack (Encoder-IP) can read and interpret the digital signals coming from the encoders. The Encoder-IP was inserted on Tech 80 carrier card. The encoders are wired directly to the encoder terminal block in Figure 4.5. The encoder terminal block connects with the Tech 80 card via a standard 50 pin ribbon cable. Subsection 4.5.3 covers wiring encoders to the encoder terminal block.

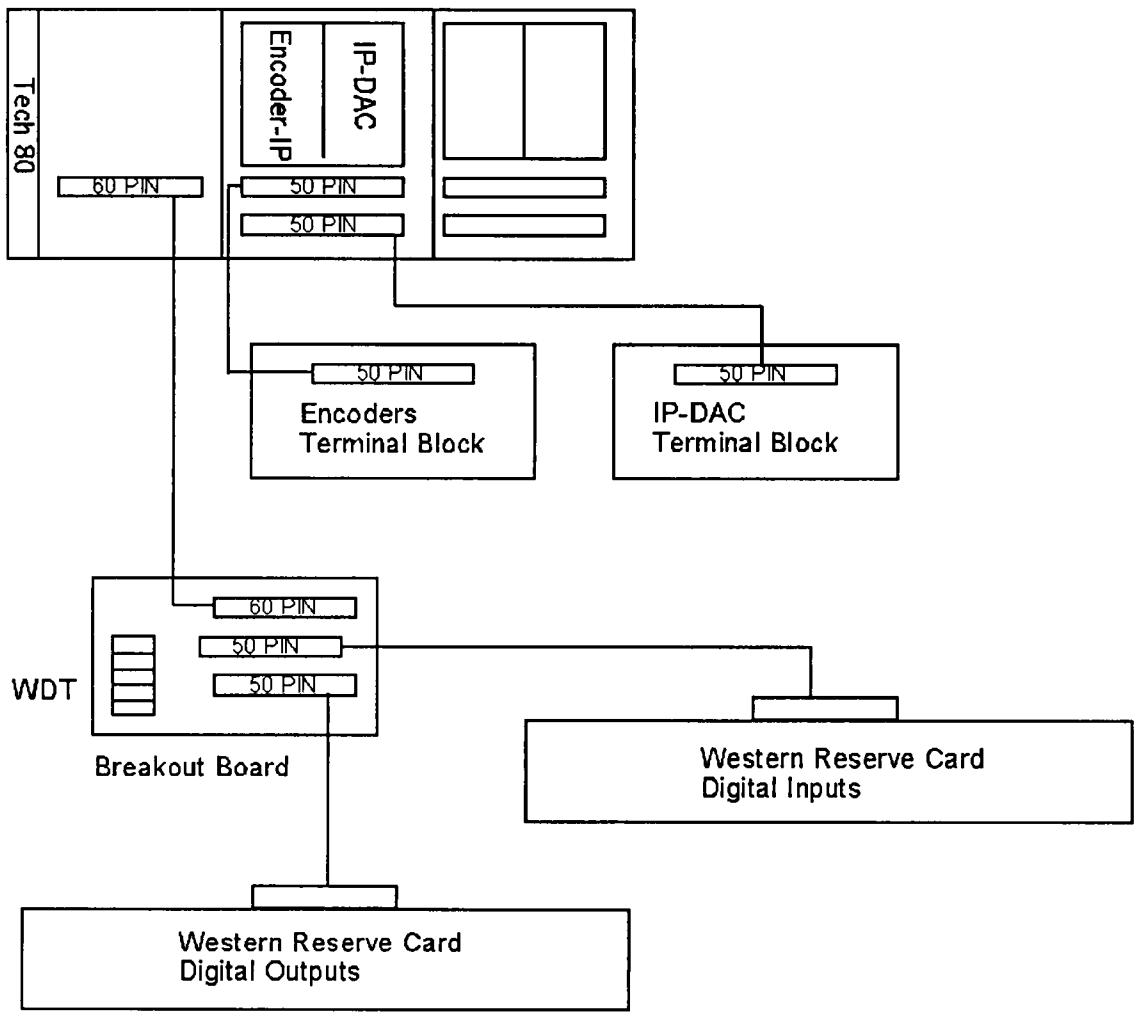


Figure 4.5: Tech 80 Industry Pack Configuration

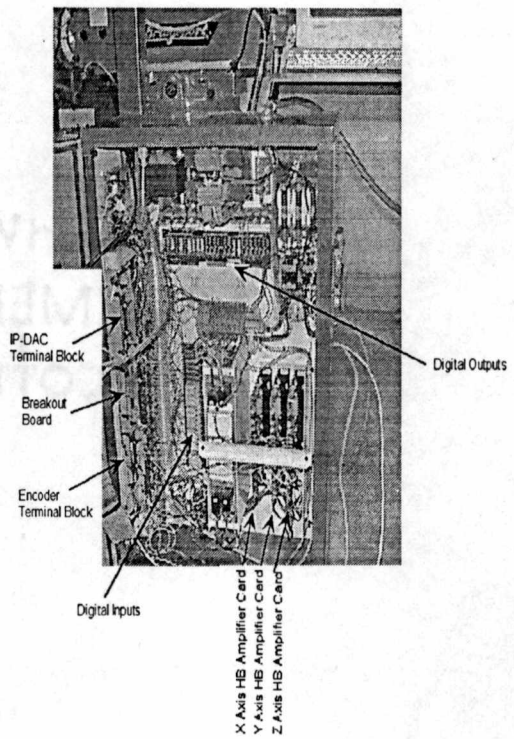


Figure 4.6: Hardware Location

4.5.1 Digital Inputs and Outputs

Digital Inputs and Outputs (DIO) are binary devices: on (closed) and off (open). When an input device does not have any type of voltage or load applied, this defines the input's normal state. The DIO for the Bridgeport was determined from the original machine prints [14] and a thorough inspection of the mill.

All of the DIO is wired to the Western Reserve Control (WRC) Card model WRC 1781-A24A to determine if an input or output is on or off. The WRC cards can monitor and control 5 VDC, 24 VDC, and 110 VAC devices. All of the digital inputs on the Bridgeport mill have voltages of 24 VDC. The digital outputs on the Bridgeport mill have voltages of 24 VDC or 110 VAC. As shown in Figure 4.7 a single WRC can receive 24 input or output modules which are numbered 0-23. These modules must be separated into groups of 8. For example if 16 inputs were desired then the modules 0-15 would need to be inputs and

modules 16-23 would be outputs. Although the inputs and outputs must be given in groups of 8, within the group modules of different voltages can exist. As long as the modules are exclusively inputs or exclusively outputs, the voltage controlled on the modules can vary. The input and output modules are polarity sensitive. For each module the odd numbered pin is the high voltage while the even pin is the lower voltage or ground. If a 24 VDC input module was inserted in slot 5 then the +24 VDC would be wired to pin 11 and the 24 VDC ground would be wired to pin 12.

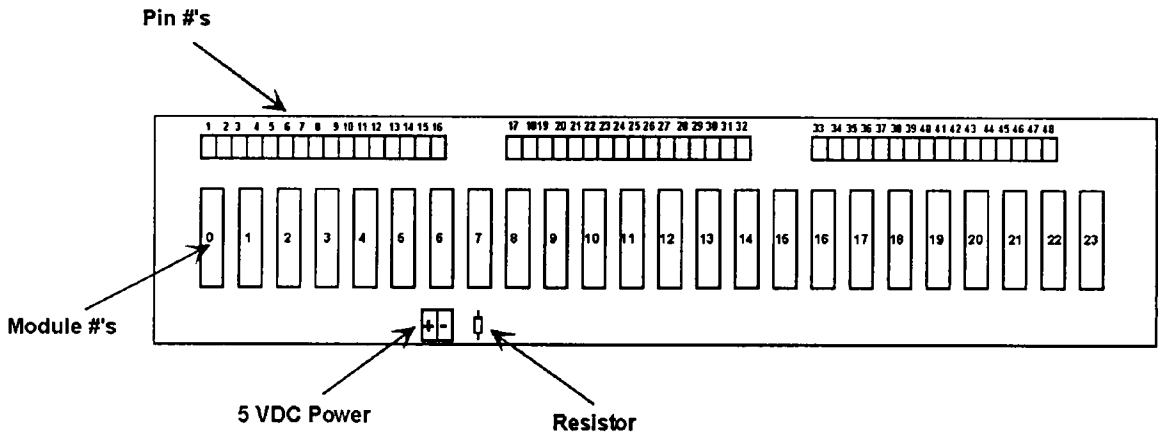


Figure 4.7: Western Reserve Card

One of the WRC cards is only used for digital inputs while the other WRC card only contains digital outputs. The Western Reserve Cards are labeled inside the Bridgeport cabinet as Digital Inputs and Digital Outputs. The resistor in Figure 4.7 must be cut off in order to use 5 VDC as the power input. Using a 5 VDC power input allows the lights on the WRC to turn on and off depending on the state of the module [13].

24 VDC Digital Inputs

Examples of Digital Inputs (DI) are limit switches, pressure switches, and level switches. The inputs can have voltages of 5 VDC or 24 VDC. These inputs are monitored using the WRC card. **DI_INPUTNAME** refers to input module monitoring the INPUTNAME. Later,

the **DI_INPUTNAMEs** will be correlated with the soft logic programs via the **dioSymsbgp1** file discussed in Subsection 4.6.5. The Bridgeport V2XT only has 24 VDC digital input which are listed in Table 4.2.

The Bridgeport V2XT has a total of 7 limit switch inputs. Limit switches determine when an axis is in a certain location. Home or reference switches define an inertial reference frame or home position so the position of the table can be ascertained in 3D space as shown in Figure 4.1. Reference switches such as X_REF, Y_REF, and Z_REF close when the axis is in the home position and have a normally open state. The limit or overtravel switches prevent the axis or slide to travel beyond its physical limits. Without the overtravel switches a machine tool could damage the table and the lead screw. Overtravel switches have a normally closed state and were inserted in the emergency stop circuit which will be discussed further in Subsection 4.5.2.

The LUBE_NOTLOW switch is a level switch for the Bridgeport's waylube container. Waylube lubricates the screw drives used to actuate the table motion. When a sufficient amount of waylube is in the container, the level switch is closed. The waylube has a normally closed condition when there is sufficient lubricant in the receptacle. The AIR_NOTLOW switch is a pressure switch to determine if there is sufficient air pressure to operate the pneumatic components of the Bridgeport. When there is enough pressure, the AIR_NOTLOW switch is closed. Therefore, the AIR_NOTLOW switch has a normally closed state when air pressure is sufficient. The SPINDLE_RDY switch is used to determine when too much torque has been applied to the motor. The SPINDLE_RDY switch opens when the torque is too high and has a normally closed state. The LOWGEAR switch determines whether the transmission for the spindle is set for low or high gear. The LOWGEAR switch is closed in low gear and open in high gear. **DI_SERVO_ENERGIZ** monitors the emergency stop circuit (ESTOP) further discussed in Subsection 4.5.2. When the ESTOP circuit is energized, the emergency stop relay (ESR) closes. When the emergency stop relay is energized, it closes a the sets of contacts necessary to supply power to the servo motors.

X_READY, Y_READY, and Z_READY monitor the HB Series Amplifiers used to amplify the position command from the computer. The HB Series Amplifier has a fault switch

Table 4.2: 24 VDC Digital Inputs on the Bridgeport V2XT Milling Machine

Input #	Input Name	Purpose	State	Device Wire #'s
0	DI_X_REF	Determine when the X Axis is in the home position	NO	103(+) 104(-)
1	DI_X_NOTOT	Determine when the X Axis slide has traveled too far in the + or - direction	NC	41(+) 42(-)
2	DI_Y_REF	Determine when the Y Axis is in the home position.	NO	105(+) 106 (-)
4	DI_Z_REF	Determine when the Z Axis is in the home position.	NO	107(+) 108(-)
5	DI_Z_NOTOT	Determine when the Z Axis slide has traveled too far in the + or - direction	NC	45(+) 47(-)
6	DI_LUBE_NOTLOW	Determine if the waylube is low	NC	101(+) 102(-)
7	DI_AIR_NOTLOW	Determine if the air pressure is low	NC	384(+) 383(-)
8	DI_SPINDLE_RDY	Determine if too much torque has been applied to the spindle	NC	63(+) 81 (-)
9	DI_LOWGEAR	Determine if the manual gear select is set for high or low gear	NC	35(+) 184(-)
10	DI_WATCHDOG_FAIL	Determine if the WDT caused an emergency stop condition	NO	N/A
11	DI_X_READY	Determine if the X Axis HB Series Amplifier is enabled	NO	480(+) 486(-)
12	DI_Y_READY	Determine if the Y Axis HB Series Amplifier is enabled	NO	480(+) 487(-)
13	DI_Z_READY	Determine if the Z Axis HB Series Amplifier is enabled	NO	480(+) 488(-)
14	DI_SERVO_ENERGIZ	Determine if ESTOP Circuit is energized to close the ESR to power the servo drives.	NO	N/A
15	DI_Y_NOTOT	Determine when the Y Axis slide has traveled too far in the + or - direction	NC	43(+) 44(-)
16	DI_CYCLE_START	Monitors the CYCLE_START switch	NO	43(+) 44(-)

to be monitored. The faults switch will open when the HB Series Amplifier has detected an error and has become disabled. X_READY, Y_READY, and Z_READY have a normally closed state when the axis is enabled and ready to send power to the servo motors. Further discussion on the HB Series Amplifier is in Subsection 4.5.4. Figure 4.8 shows the 24 VDC digital inputs wiring diagram.

CYCLE_START is a hard wired push button to start a part program execution and **DI_CYCLE_START** monitors its condition. CYCLE_START has a normally open state.

24 VDC Digital Outputs

The 24 VDC Digital Outputs (DO) are devices such as solenoids and relays as shown in Table 4.3 and Figure 4.9. **DO_OUTPUTNAME** refers to the output module on the WRC card controlling OUTPUTNAME. Later, the **DO_OUTPUTNAMEs** will be correlated with the soft logic programs used to control the milling machine.

The NOT_BRAKE solenoid is used to actuate the pneumatic brake to stop the spindle from turning. The Bridgeport's spindle is driven by an Emerson Electric three phase AC motor. Since an AC three phase motor is configured for a constant speed, the spindle speed is adjusted by a variable speed transmission. The transmission speed reduction ratio can be changed by adjusting a variable pitch pulley with a pneumatic actuator. The SSINC solenoid activates a pneumatic actuator to adjust the transmission speed ratio so that spindle speed is increased. Likewise, the SSDEC solenoid activates a pneumatic actuator to reduce spindle speed. In order to change the spindle turning direction, the three phase motor's line voltage polarity must be reversed. The SPINDLE_CW relay closes a set of contacts to the three phase motor causing the spindle to turn forward or clockwise in low gear. The SPINDLE_CCW relay closes a separate set of contacts to reverse the polarity and turn the spindle in the counter clockwise direction in low gear. The opposite is true in high gear. Closing the SPINDLE_CW contacts in high gear turns the spindle counter clockwise. Closing the SPINDLE_CCW contacts in high gear turns the spindle clockwise. **DO_COOLNT_MIST** controls a solenoid to spray mist coolant on the part during a cutting procedure. The COOLNT_MIST solenoid is not installed currently, but the wiring and

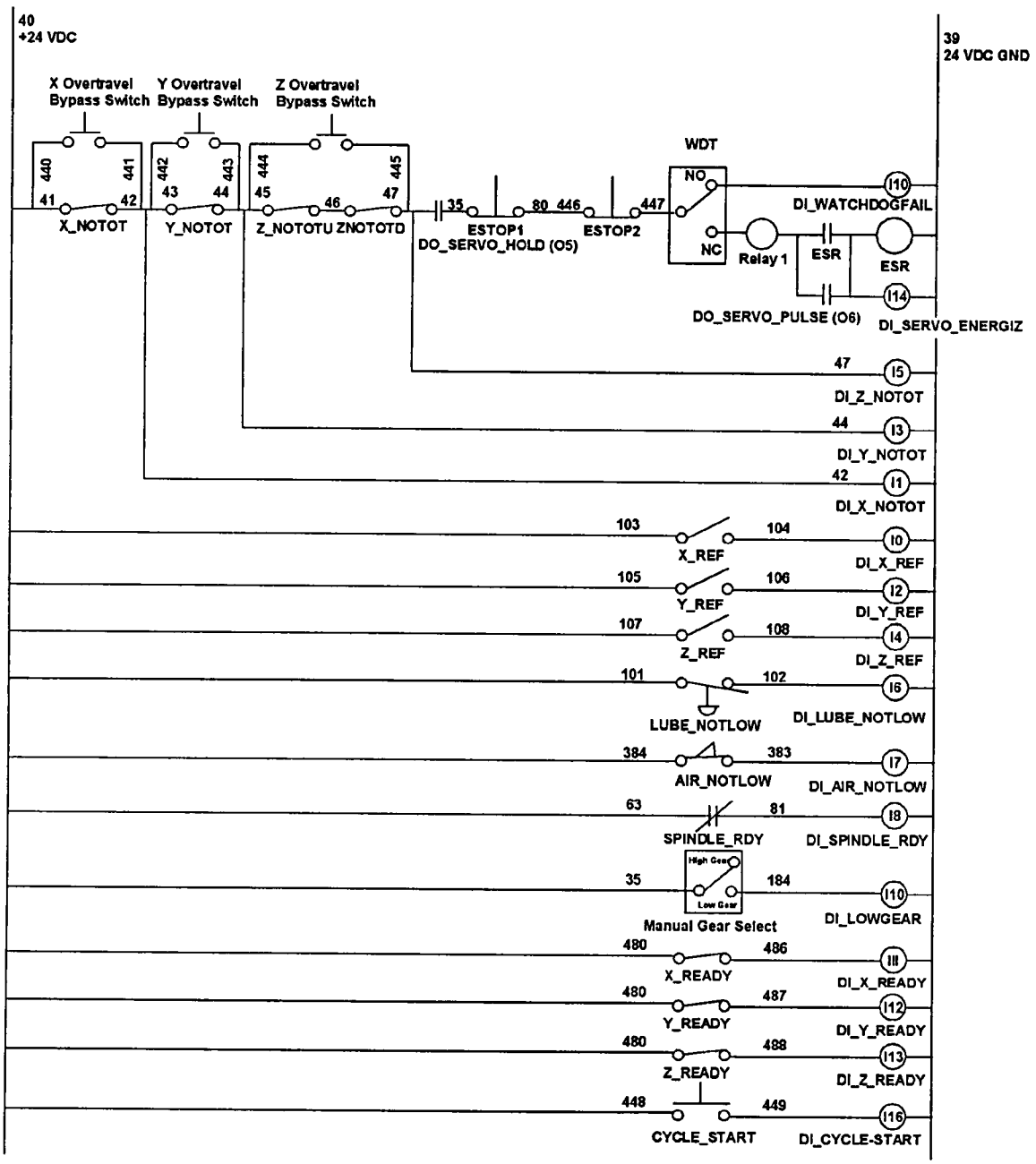


Figure 4.8: 24 VDC Digital Input Wiring Diagram

Table 4.3: 24 VDC Outputs on the Bridgeport V2XT Milling Machine

Output #	Output Name	Device Type	Purpose	Device Wire #'s
0	DO_SSINC	Solenoid	Activate a pneumatic actuator to increase spindle speed	118(+) 117(-)
1	DO_SSDEC	Solenoid	Activate a pneumatic actuator to decrease spindle speed	116(+) 115(-)
2	DO_NOT_BRAKE	Solenoid	Activate the pneumatic brake to stop the spindle	61(+) 60(-)
4	DO_SPINDLE_CW	Relay	Close a set of contacts to spin the spindle clockwise	57(+) 59(-)
6	DO_SERVO_PULSE	Relay	A momentary contact to initially energize the ESTOP circuit.	
7	DO_SPINDLE_CCW	Relay	Close a set of contacts to spin the spindle counter clockwise	54(+) 56(-)
8	DO_SERVO_HOLD	Relay	A software variable to deenergize the ESTOP circuit in the event of a software error.	
9	DO_COOLNT_MIST	Solenoid	Activate a solenoid to provide mist coolant while cutting a part. Not currently installed.	N/A

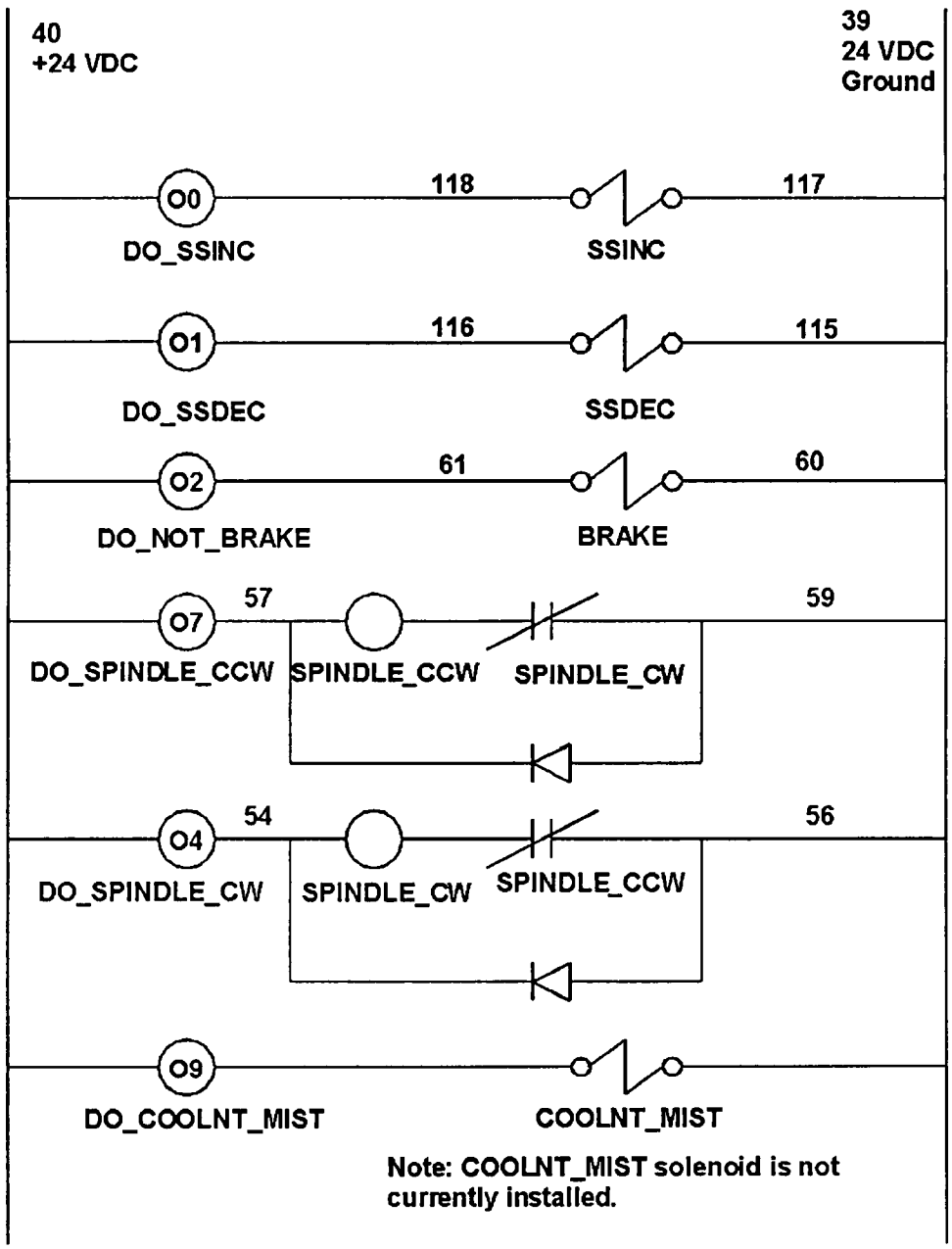


Figure 4.9: 24 VDC Digital Output Wiring Diagram

softlogic are ready to support a COOLNT_MIST solenoid. The mist coolant is manually turned on by the operator when cutting a part.

110 VAC Digital Outputs

Figure 4.10 shows the wiring of the 110 VAC outputs. The 110 VAC outputs are a set of contacts to control a variety of devices such as solenoids and relays. Table 4.4 shows the 110 VAC outputs. **DO_LUBE_PMP** closes a set of contacts to give power to the lube pump. LUBE_PMP lubricates the slides with waylube in order to reduce friction. **DO_COOLNT_SPRAY** activates a 125 VAC receptacle on the back of the Bridgeport control cabinet. If a coolant pump was add later to provide flood coolant, it would simply be mounted and plugged into the back of the Bridgeport cabinetry. The wiring and softlogic already support a coolant pump which can be added at a later time.

5 VDC Wiring

Although 5 VDC is not used to power any digital inputs or outputs, it does power the encoders and the Western Reserve Control cards as shown in Figure 4.11.

4.5.2 Emergency Stop Circuit

The emergency stop circuit (ESTOP) is a 24 VDC circuit to stop the machine tool operation in the event of an emergency. The ESTOP is a series circuit containing hardware with normally closed contacts. If one of the contacts is opened, the circuit will lose power and cause the emergency stop relay (ESR) to lose power. When the emergency stop relay is energized, the spindle and servo drives receive power. During an emergency condition the ESR will lose power, thereby removing power to the servo drives and the spindle. The ESTOP was designed to stop the machine tool without the aid of a computer hardware or software. Figure 4.12 shows the ESTOP circuit for the Bridgeport milling machine.

X_NOTOT, Y_NOTOT, Z_NOTOTU, and Z_NOTOTD are the overtravel switches to stop the servo drives from driving the table off of the drive screws. The ESTOP1 and

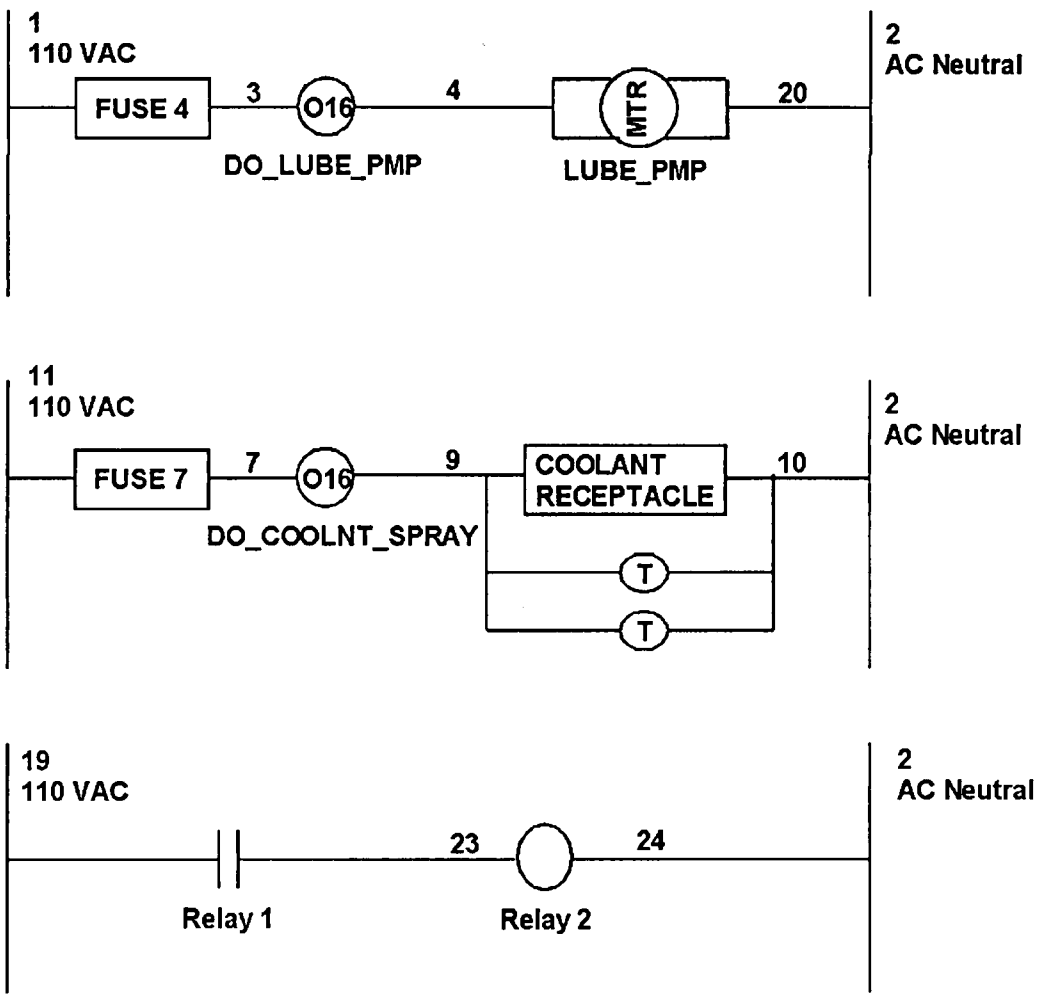


Figure 4.10: 110 VAC Digital Output Wiring Diagram

Table 4.4: 110 VAC Outputs on the Bridgeport V2XT Milling Machine

Output #	Output Name	Device Type	Purpose	Device Wire #'s
16	DO_LUBE_PMP	Solenoid	Activate the 110 VAC LUBEPUMP to supply waylube to actuator slides.	4(+) 20(-)
17	DO_COOLNT_SPRAY	Coolant Pump	Activate a 125 VDC receptacle on the back of the Bridgeport cabinetry. When a coolant pump was plugged into the receptacle, the pump would supply flood coolant.	9(+) 10(-)

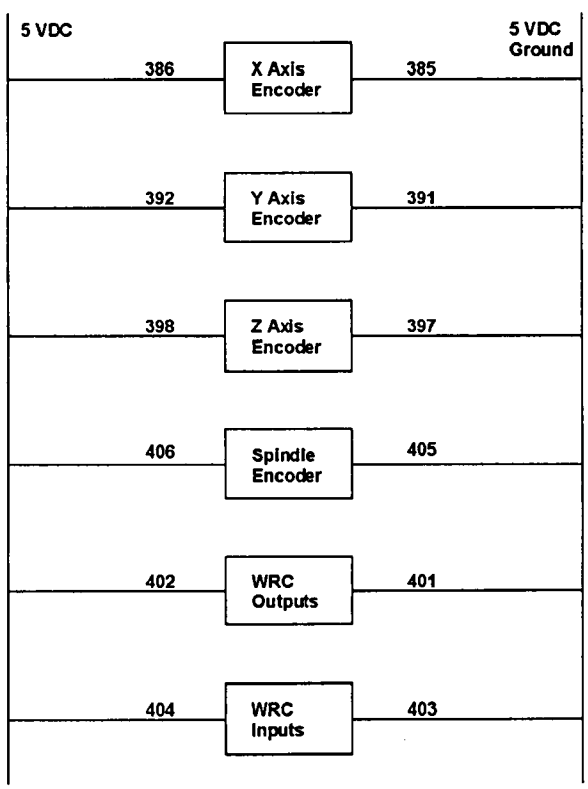


Figure 4.11: 5 VDC Wiring Diagram

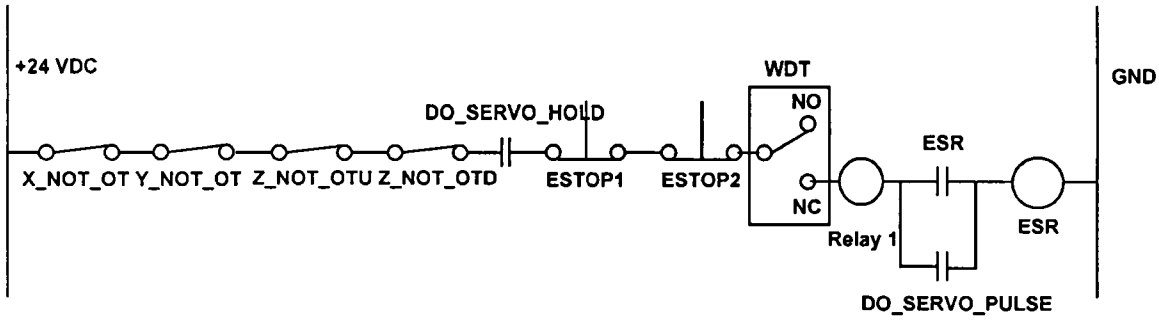


Figure 4.12: Emergency Stop Circuit

ESTOP2 are large red pushbuttons switches. These switches are strategically placed on the machine tool so an operator can stop the machine quickly. The watch dog timer (WDT) is a timer that must be reset by the hardware every 2 milliseconds. The WDT timer is a safety switch to ensure the digital systems are performing normally. **DO_SERVO_HOLD** is a software variable that controls a digital output in the ESTOP circuit. In the event of a serious software error, **DO_SERVO_HOLD** will open and stop the machine tool. **DO_SERVO_PULSE** is a momentary contact that will initially energize the ESR. After the ESR has been initially energized, **DO_SERVO_PULSE** will open.

4.5.3 Encoder Industry Pack

Digital encoders are a used to determine the position of the axes. For this application an encoder is also being used to determine spindle speed. The encoders on the Bridgeport are two-channel quadrature type which produce a square wave output. The Model 12-6 Encoder Industry Pack [12] can support up to six channels or encoders. The pin assignments for the Encoder Industry Pack are shown in Table 4.5. There are two channels given in Table 4.5 . The channel given in parentheses is the channel number used by OpenCNC^{TM} . The channel number without parentheses is the channel number used in the Encoder-IP documentation [12]. From [14] the encoders for axes X, Y, and Z have 250 lines or 1000 pulses per revolution. The spindle encoder has 1024 lines or 4096 pulses per revolution. Figure 4.13 shows the wiring for all the encoders to the encoder terminal block in the Bridgeport cabinetry.

Table 4.5: Pin Assignments for the Tech 80 Model 12 Encoder Module

Pin	Channel	Function	Pin	Channel	Function
1	GND	Ground	26	GND	Ground
2	6 (0)	+Index	27	3 (3)	+Index
3	6 (0)	-Index	28	3 (3)	-Index
4	6 (0)	+B	29	3 (3)	+B
5	6 (0)	-B	30	3 (3)	-B
6	GND	Ground	31	GND	Ground
7	6 (0)	+A	32	3 (3)	+A
8	6 (0)	-A	33	3 (3)	-A
9	GND	Ground	34	GND	Ground
10	5 (1)	+Index	35	2 (4)	+Index
11	5 (1)	-Index	36	2 (4)	-Index
12	5 (1)	+B	37	2 (4)	+B
13	5 (1)	-B	38	2 (4)	-B
14	GND	Ground	39	GND	Ground
15	5 (1)	+A	40	2 (4)	+A
16	5 (1)	-A	41	2 (4)	-A
17	GND	Ground	42	GND	Ground
18	4 (2)	+Index	43	1 (5)	+Index
19	4 (2)	-Index	44	1 (5)	-Index
20	4 (2)	+B	45	1 (5)	+B
21	4 (2)	-B	46	1 (5)	-B
22	GND	Ground	47	GND	Ground
23	4 (2)	+A	48	1 (5)	+A
24	4 (2)	-A	49	1 (5)	-A
25	GND	Ground	50	GND	Ground

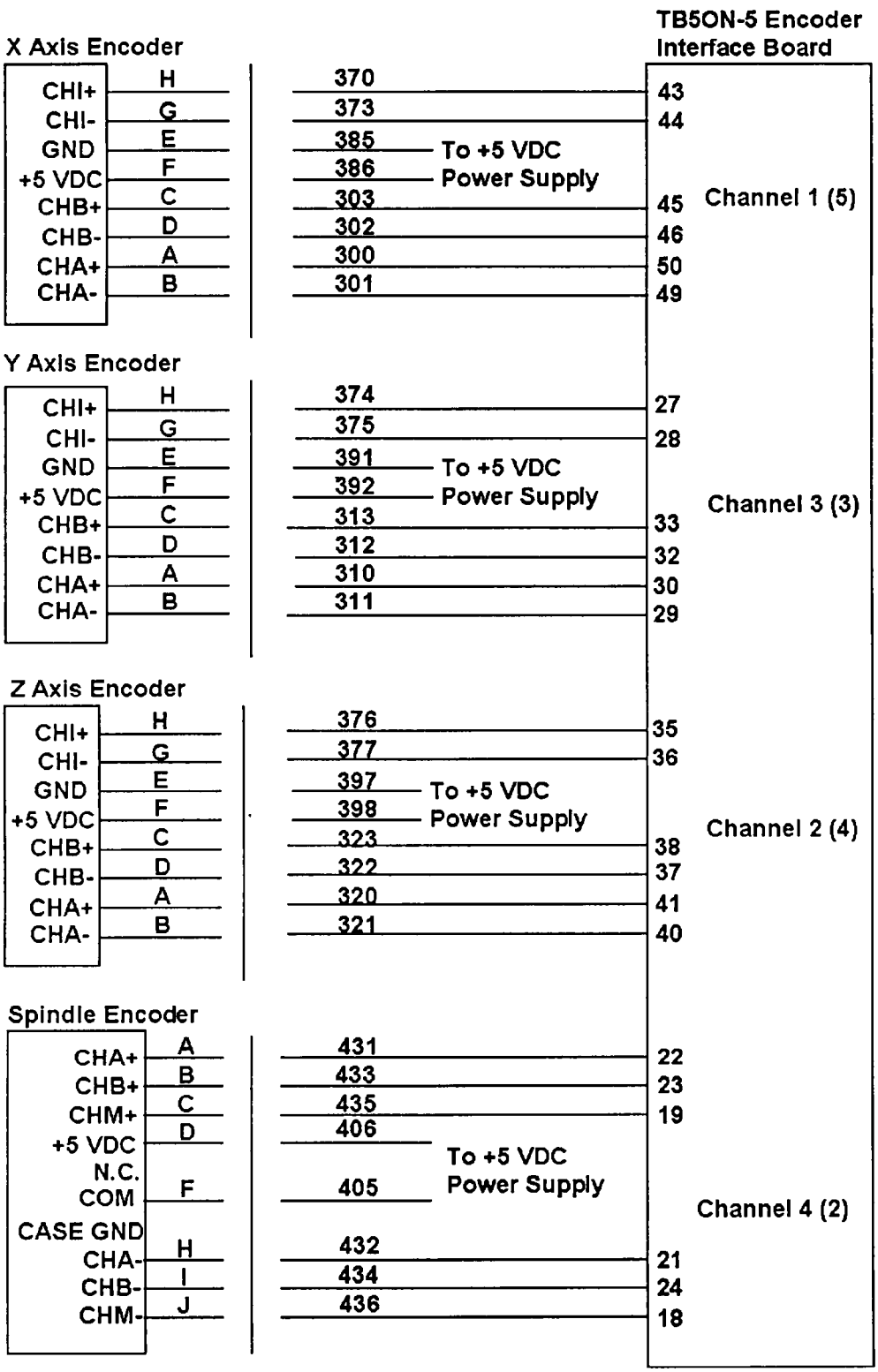


Figure 4.13: Encoder Wiring Diagram

4.5.4 DAC Industry Pack

The Greenspring IP-DAC Industry Pack has six channels to perform digital to analog convertors [16]. The IP-DAC is used to send an analog velocity command to the HB Series Amplifier. Table 4.6 shows the pin assignments for the IP-DAC terminal block in the Bridgeport cabinetry. There are two channels given in Table 4.6 . The channel given in parentheses is the channel number used by OpenCNC^{TM} . The channel number without parentheses is the channel number used in the IP-DAC documentation [16]. Most DC servo motor amplifiers receive a -10 to +10 VDC command input voltage. The IP-DAC was jumper configured to produce a signal ranging from -10 to +10 VDC for all of the channels. In the event of motor and amplifier replacement, the IP-DAC may need to be reconfigured. Consult [16] for IP-DAC voltage output ranges and jumper configuration.

The three motion axes on the Bridgeport mill are controlled by the HB Series Amplifier from the Motion Science Corporation of Palo Alto, California. The HB Amplifier provides the servo level control for the individual axes. Each axis has its own HB Series Amplifier to control its motion. The HB Series Amplifier has two terminal strips J1 and J2 that receive inputs from the external control system as shown in Figure 4.14.

The J1 terminal block receives the voltage command from the control computer, the feedback from the motor tachometer, and the commands involving fault shutdown of the servo drives. Table 4.7 defines the functions of the individual pins for the J1 terminal block. The J2 terminal strip receives power from the mill's power supplies and powers the DC servo motors. Table 4.8 cites the functions of the individual pins for the J2 terminal block. The HB Series amplifier has current limit, current gain, tachometer scaling, and balance adjustments. The balance adjustment was used to null out drift in the axes due to noise from the IP-DAC. The other adjustments were left at the settings tuned in by Bridgeport during manufacturing.

Table 4.6: Pin Assignments for the IP-DAC

Pin	Function	Pin	Function
1	Analog Ground	26	DAC 5 (4) Output
2	DAC 1 (0) Output	27	DAC 5 (4) Sense
3	DAC 1 (0) Sense	28	Analog Ground
4	DAC 1 (0) Current In	29	n/c
5	n/c	30	n/c
6	DAC 1 (0) Current Out	31	Analog Ground
7	Analog Ground	32	DAC 6 (5) Output
8	DAC 2 (1) Output	33	DAC 6 (5) Sense
9	DAC 2 (1) Sense	34	Analog Ground
10	DAC 2 (1) Current In	35	n/c
11	n/c	36	n/c
12	DAC 2 (1) Current Out	37	Analog Ground
13	Analog Ground	38	n/c
14	DAC 3 (2) Output	39	n/c
15	DAC 3 (2) Sense	40	Analog Ground
16	Analog Ground	41	n/c
17	n/c	42	n/c
18	n/c	43	Analog Ground
19	Analog Ground	44	n/c
20	DAC 4 (3) Output	45	n/c
21	DAC 4 (3) Sense	46	Analog Ground
22	Analog Ground	47	n/c
23	n/c	48	Analog Ground
24	n/c	49	+5.000 VDC Drive Input/Output
25	Analog Ground	50	n/c

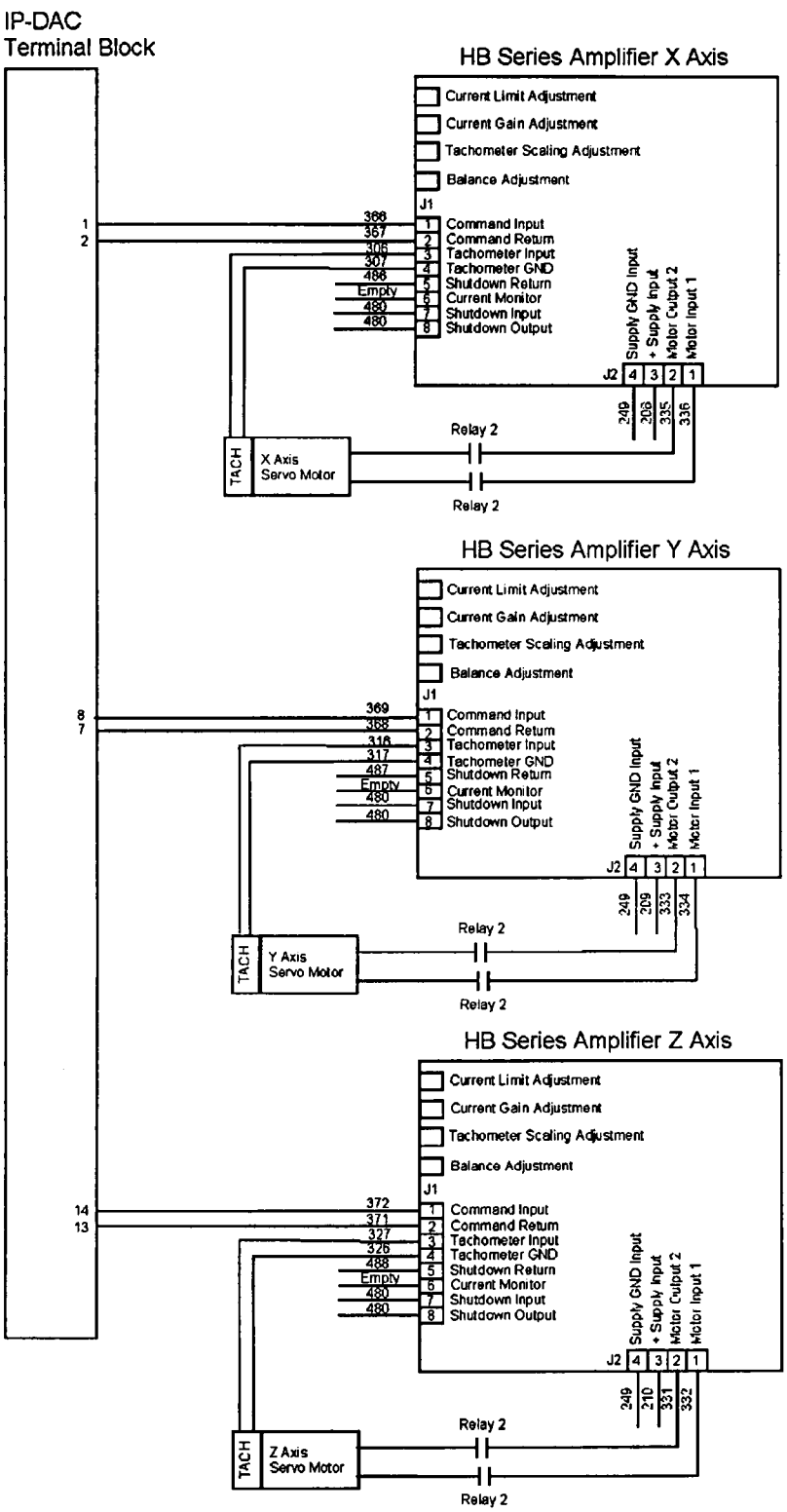


Figure 4.14: IP-DAC Wiring Diagram

Table 4.7: HB Series Amplifier J1 Terminal Strip Pin Assignments

J1 Pin	Name	Description
1	Command Input	The Command Input is the analog command voltage from the control computer. This voltage ranges from +/- 10 VDC with respect to the Command Return (J2 Pin2). A + 10 VDC should drive the servo at maximum velocity in the positive direction. If this does not occur the signal can be adjusted using the Current Gain (CG) or the Tachometer Scaling (VG) adjustments.
2	Command Return	The Command Return is the ground for the Command Input.
3	Tachometer Input	The Tachometer Input is an analog voltage from the tachometer attached to the DC motor.
4	Tachometer GND	The Tachometer Ground is the ground return for the Tachometer Input (J1 Pin 3).
5	Shutdown Return	Shutdown Return is the ground return for the Shutdown Input (J1 Pin7) and Shutdown Output (J1 Pin8).
6	Current Monitor	Current Monitor is a bipolar analog output signal which is proportional to amplifier output current on J2, pins 1 and 2, the Motor Outputs.
7	Shutdown Input	Shutdown Input is a method for the control computer to shutdown the Motor Outputs on J2 when it is active. The condition of the Shutdown Input is determined by jumper JP4. When JP4 is in, the a voltage must be applied in order to enable the servo drive. Shutdown Input is compatible with 5 to 30 VDC logic. The JP4 jumper is in on all the HB Series Amplifiers for the Bridgeport mill.
8	Shutdown Output	Shutdown Output is a method for the control computer to know when the HB Series Amplifier has shutdown power to the Motor Outputs on J2. The condition of the Shutdown Output is determined by jumper JP5. When JP5 is out, an error is shown as an open condition. The JP5 jumper is out on all the HB Series Amplifiers for the Bridgeport mill.

Table 4.8: HB Series Amplifier J2 Terminal Strip Pin Assignments

J2 Pin	Name	Description
1	Motor Output 1	Motor Output 1 is one of two power signals sent to the DC Motor. Motor Output 1 is a 22 KHz pulse width modulated signal that switches between the supply ground input (Pin4) and the + supply voltage (Pin 3). Motor Output 1 has a (+/-) differential polarity with respect to the J1 Command Input.
2	Motor Output 2	Motor Output 2 is the other power signal sent to the DC Servo Motor. Motor Output 2 is a 22 KHz pulse width modulated signal that is identical to Motor Output 1 except it switches at the opposite times. Motor Output 1 has a (-/+) differential polarity with respect to the J1 Command Input.
3	+ Supply Input	+ Supply Input receives the positive voltage from the external power supply. The Supply Input receives the power to supply the HB Amplifier and the DC motors.
4	Supply GND Input	Supply GND Input is the ground from the external power supply.

4.6 Software Configuration

OpenCNC^{TM} requires the use of two operating systems to develop the control and operate the Bridgeport mill . OpenCNC^{TM} uses the QNX^{TM} operating system for PCs to run OpenCNC^{TM} in conjuction with the soft logic developed specifically to control the mill. ISaGRAF^{TM} , the commercially available soft logic tool, runs on the Windows 95^{TM} operating system to develop the soft logic used in conjunction with OpenCNC^{TM} .

4.6.1 Operating Systems

The server and the client run both the QNX^{TM} and the Windows 95^{TM} operating systems. Both computers came originally loaded with Windows 95^{TM} . The QNX^{TM} operating system and OpenCNC^{TM} were installed using the instructions in [15]. During the installation process two 500 megabyte (MB) QNX^{TM} partitions were created. Having two QNX^{TM} partitions allows for redundancy in operating the computer. If software changes or upgrades

are needed, one partition could be edited while the other partition remains intact. Since the server and the client each have two QNX partitions, the mill can be operated by four different QNX^{TM} partitions.

The QNX^{TM} operating system has two accounts: root and bridge. root is the system administrator or superuser account on the QNX operating system. bridge is the operator account to run the machine tool. The bridge account was created by typing the following text at the command while logged in as root:

passwd bridge

This command will create the bridge account and create a **/home/bridge** directory with a **.profile file.** A **.profile** file contains the paths to files and directories containing the bridge control software.

4.6.2 Network Configuration

The Manufacturing Systems Laboratory contains a server and a client. Part programs, the soft logic, and the QNX^{TM} operating system are down loaded over the Ethernet from the server to be run on the client. The server and client contain NE 2000 compatible Ethernet cards with space for MDSI's boot read only memory (BROM) chip. The MDSI BROM was inserted on to the client and causes the client to request to be booted from the server. If the server does not respond, the client can be booted up using its own hard disk.

Certain files in the **/etc/config** directory need to be created and owned by root on the server in order to boot the client from the server. The **netmap** file contains the node number, the local area network number, and the Ethernet address of each computer as shown in Table 4.9. The node number is used by OpenCNC^{TM} to designate a computer. The server is node 1 and the client is node 2. Any other machine tools added at a later time should follow sequentially after 2. The local area network (LAN) is 1. Each Ethernet card is given a unique address during manufacturing. These Ethernet addresses allow the server and the client to recognize one another.

The sysinit files are used to start the device drivers and system software drivers for the computers. **sysinit.1** is the sysinit file for the server and sysinit.2 is the sysinit file for the

Table 4.9: /etc/config/netmap

```
# netmap contains the information necessary to boot the Bridgeport
# Mill Client from the server
# Node LAN Ethernet Address Computer
1 1 0000B4 83016B # Server
2 1 0000B4 82F56B # Bridgeport Mill Client
```

Table 4.10: Ethernet Cards' Information

computer	Ehternet Address	I/O BASE	IRQ	BROM BASE SIZE
server	0000B4 83016B	0x240	9	
client	0000B4 82F56B	0x320	5	CC00:0000

client. The suffix .1 and .2 refer to the node number of the computer. The /etc/config directory contains examples of the following files: **sysinit.server**, **sysinit.client_w_disk**, **sysinit_wo_hardisk**, and **sysinit.standalone**. The QNX copy command (cp) was used to copy one of the files to its new filename while leaving the example intact. To copy **sysinit.server** to **sysinit.1** the syntax is

cp sysinit.server sysinit.1

The interrupt requests (IRQ), input/output base addresses (I/O BASE), and the BROM base sizes (BROM BASE SIZE) must be set within the sysinit.1 and sysinit.2 files. Table 4.10 shows the IRQ's, I/O BASE, and BROM BASE SIZES for the server and the client. Copies of **sysinit.1** and **sysinit.2** are located in Appendices D and E.

4.6.3 ISaGRAF^{TM}

OpenCNC^{TM} uses CJ International's ISaGRAF^{TM} [11] to develop the soft logic. Two projects were developed within ISaGRAF^{TM} to run the Bridgeport milling machine: **bgp1dio** and **bgp1msg**.

The **bgp1dio** project controls the digital inputs and outputs on the machine tool. The

contents of **bgp1dio** project are shown in Figure 4.15. Each project contains three sections: begin, sequential, end, and functions. The client computer executes the begin section, the sequential section, and finally the end section. The begin section initializes the software to control the machine tool. For example **init** attaches a series of OpenCNC^{TM} arrays for use by **bgp1dio**. The sequential section runs a set of sequential function charts. A sequential function chart (SFC) is a graphically based programming language that executes in a sequential order [11]. **runInChg** and **runTime** SFC's check for machine and software faults. **runEnerg** is a sequential function chart used to initially energize the ESTOP circuit. **runReset** and **runSync** are sequential function charts to sync or home the mill in order to establish an inertial reference frame. The end section contains only one program setState which updates various variables used by the graphical user interface. The function section contains programs called by the other three sections. The **setSpnSt** program controls the spindle. The **setCooSt** program controls the automatic use of coolant.

Figure 4.16 shows the contents of the bgp1msg project. The **bgp1msg** project handles the messages sent to operator's screen when running the machine tool. The **bgp1msg** present warnings and messages depending on the functions requested by the operator or the part program. The structure is similar to **bgp1dio**.

4.6.4 /home/bridge

The **/home/bridge** is the home directory for the bridge operator account and contains the following files and directories:

1. **.profile**
2. **runbridge**
3. **/program**
4. **/offset**
5. **/fixture**
6. **/windows**

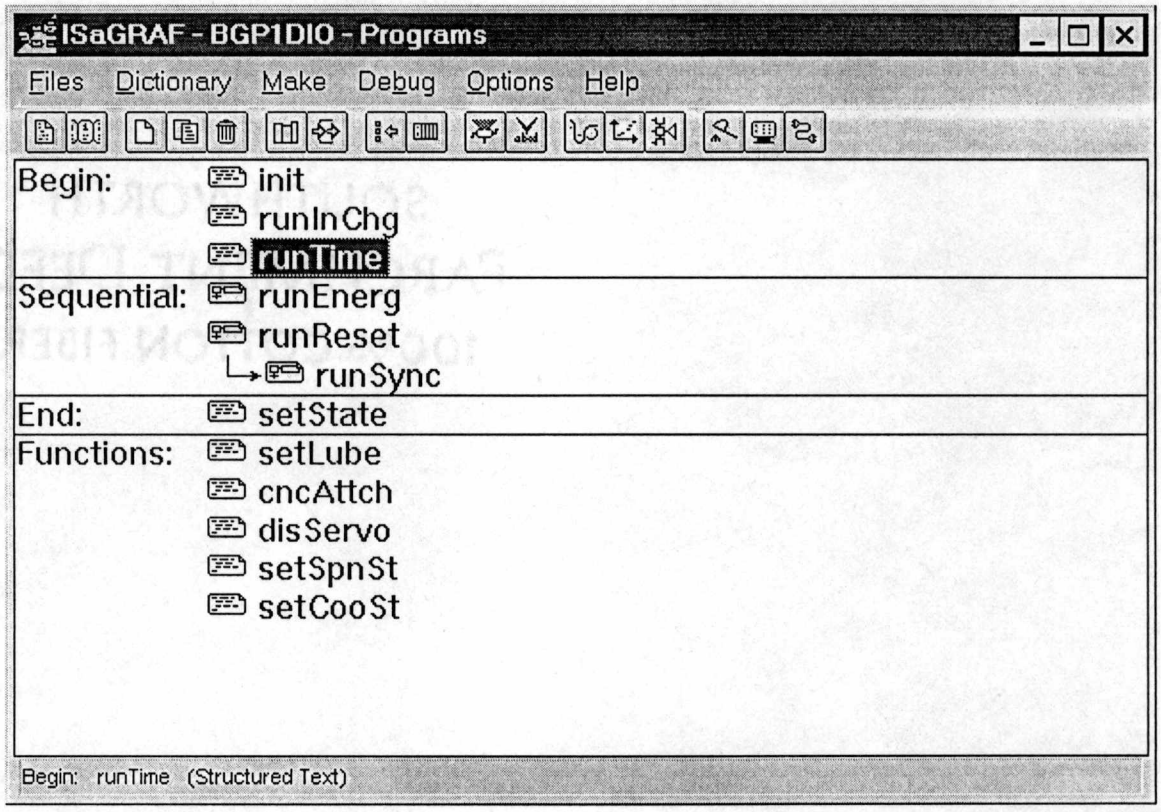


Figure 4.15: ISaGRAF^{TM} bgp1dio project

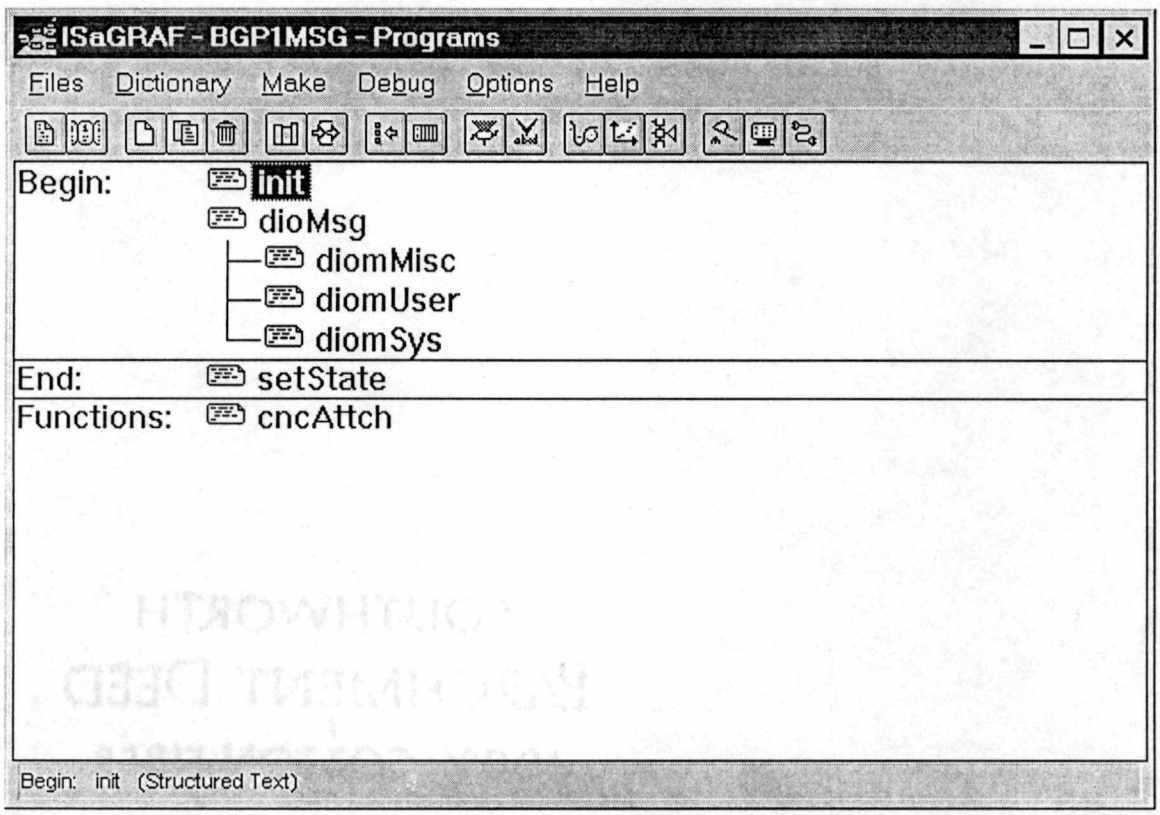


Figure 4.16: ISaGRAF bgp1msg project

Table 4.11: runbridge

```
windows
winMenu -m 1 -f /usr/mdsi/2.0/machines/bridge/bgp1.men
```

The **.profile** is created by QNX^{TM} when setting up the bridge account [15] and contains paths to files and directories containing the bridge control software. The complete **.profile** file is given in Appendix F. The **runbridge** file contains two lines of code as shown in Table 4.11. An operator logs in as bridge and types runbridge to start the machine tool. The windows command starts QNX^{TM} windows to provide the infrastructure to build the graphical user interface. The second line starts the Bridgeport mill custom menu file to provide the operator with the graphical user interface. **runbridge** should be owned by the bridge account. Two lines of code are necessary to create the **runbridge** file, insure proper privledges to the users, and establish bridge account ownership of **runbridge**:

vedit runbridge

chmod a+w runbridge

chown=bridge

The **/program** directory stores the part programs to be run by the operator. The **/offset** directory contains the offsets due to different tools used on the Bridgeport mill. The /fixture directory contains locations of fixtures on the Bridgeport mill. Using the **/program** directory as an example, the **/program**, **/offset**, and **/fixture** were created, given appropriate privileges, and established bridge as the owner of these directories with the following lines of code:

mkdir /program

chmod a+w /program

chown -R bridge:100 /program

The **/windows** directory was created the first time the bridge account started QNX^{TM}

windows.

4.6.5 /usr/mdsi/2.0/machines/bridge

The **/usr/mdsi/2.0/machines/bridge** directory contains all of the files necessary to run the Bridgeport mill in conjunction with OpenCNC. The **/usr/mdsi/2.0/machines/bridge** directory contains the following files:

1. **startbgp1**
2. **dioSymsbgp1**
3. **tunebgp1**
4. **bgp1.gen**
5. **bgp1.men**
6. **bgp1dio.11**
7. **bgp1dio.16**
8. **bgp1dio.26**
9. **bgp1dio.dco**
10. **bgp1dio.dlo**
11. **bgp1msg.21**
12. **bgp1msg.dco**
13. **bgp1dio.dlo**

startbgp1

startbgp1 is the start up script that reads all of the files in the **/usr/mdsi/2.0/machines/bridge** directory as well as the necessary OpenCNC^{TM} files to run the Bridgeport mill. If a message at start up includes, "Cannot find /directory/filename", the **startbgp1** probably has an incorrect pathname to a file. A complete listing with comments of **startbgp1** is in Appendix G.

dioSymsbgp1

The **dioSymsbgp1** maps the DIO in Subsection 4.5.1 the WRC cards to be used by OpenCNC and the soft logic developed in ISaGRAF^{TM} . The inputs are given first followed by the outputs. The inputs and outputs names should be stated in sequential order corresponding to hardware numbers on the WRC cards. The names given to the DIO in the **dioSymsbgp1** are used in the ISaGRAF^{TM} softlogic to monitor and control the DIO. Appendix H contains a hard copy of **dioSymsbgp1.**

tunebgp1

The **tunebgp1** or tune file defines important setup variables to be interpreted by OpenCNC^{TM} . A complete listing of **tunebgp1** is located in Appendix I. This section will discuss the key variables in the **tunebgp1** file. A complete listing of all OpenCNC^{TM} variables can be found in [17].

defAxisCount, **jsAxisBase**, and **defAxisLetters** define the axes and their job stream. A job stream is used by OpenCNC^{TM} to control the flow of information. **defAxisCount** defines the number of axes not counting the spindle. **jsAxisCount** defines the number of axes in job stream. The Bridgeport mill contains only job stream 0 with all three axes controlled by job stream 0. **jsAxisBase** defines the first axis in job stream 0. In this case the first axis in the job stream would be the Axis 0 or X. **defAxisLetters** correlates the axis letter with its axis number. For example

defAxisLetters XYZ

would correlate X, Y, and Z to axis numbers 0, 1, and 2.

dacRemap[] and **encRemap[]** correlate the channels coming for the Encoder-IP and IP-DAC to their respective axis number. The **dacRemap[]** variable defines which IP-DAC channel corresponds to which axis. For example

dacRemap[1] 0

would map IP-DAC channel 0 to axis 1. The **encRemap[]** variable defines which encoder channel correspond to each axis. For example

encRemap[1] 0

would map Encoder-IP channel 0 to axis 1.

axFErrGain[], **defAccelMax**, and **defJerkMax** are the variables used to tune the axis motion control. **axFErrGain[]** is a linear gain that scales the position error. Each axis has its own **axFErrGain[]** variable. For example

axFErrGain[0] 20000

would set the position error gain to 20000 for axis 0. **defAccelMax** defines the maximum allowable acceleration for all of the axes. **defJerkMax** defines the maximum allowable jerk for all of the axes.

bgp1.gen

The **bgp1.gen** files creates the graphical user interface for the Bridgeport mill control. **bgp1.gen** builds all of the buttons and pull down menus for the GUI. A complete listing of **bgp1.gen** is listed in Appendix J.

bgp1.men

The **bgp1.men** creates the pull down menu for the MDSI button in the top right corner of the QNX^{TM} windows screen. A complete listing of the **bgp1.men** is listed in Appendix K The numbers on the far left in the **bgp1.men** define authorization level necessary to use that particular function. The highest level of authorization for the Bridgeport system is 3. Consult Appendix B for more information on changing the authorization level.

bgp1dio and bgp1msg

The **bgp1dio** and **bgp1msg** are the digital input/output and message projects created in ISaGRAF^{TM} on the Windows 95^{TM} partition. All the **bgp1dio** and **bgp1msg** files were moved from the Windows 95^{TM} partition to the QNX^{TM} partition by typing the following from the **/usr/mdsi/2.0/machines/bridge directory:**

/usr/mdsi/2.0/bin/getIsa bgp1dio bgp1msg

Chapter 5

PC Based Computer Aided Manufacturing

Due to the development of several computer aided design (CAD) and computer aided manufacturing (CAM) software, engineers can design parts, and produce part programs from their PC. Because many software products focus on either CAD or CAM software, these companies have cooperated in order to exchange files. This chapter is devoted to exploring the capabilities of SolidWorks97Plus^{TM} and SURFCAM^{TM} . SolidWorks97Plus^{TM} is a three dimensional CAD package developed by the SolidWorks Corporation. SURFCAM^{TM} is a CAM package, developed by SURFWARE Inc., used to simulate machining on various CNC machine tools and to produce actual part programs for part production. Figure 5.1 shows the engineering design process using SolidWorks97Plus^{TM} and SURFCAM^{TM} .

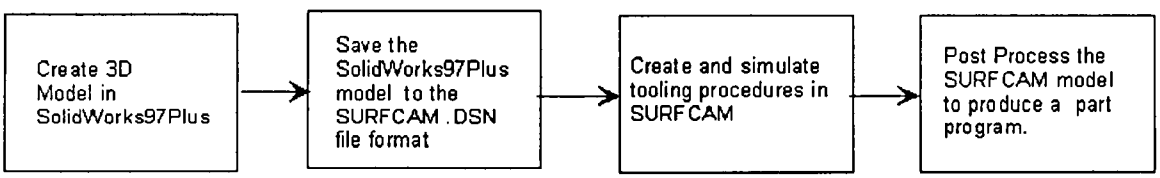


Figure 5.1: Design Process

The parts will be designed in SolidWorks97Plus^{TM} and then saved to the DSN format used by SURFCAM^{TM} . Once a part is opened in SURFCAM^{TM} , tooling operations will be chosen in order cut the part from a piece of stock. Finally, the post processor in SURFCAM^{TM} will be used to produce a part program. In order to illustrate the capabilities

of SolidWorks97Plus^{TM} , SURFCAM^{TM} , and OpenCNC^{TM} , a bearing bracket was chosen as a demonstration part. This chapter will cover the design, simulation, and production of the bearing bracket using these CAD and CAM tools.

5.1 SolidWorks97Plus

SolidWorks97Plus^{TM} allows the designer to sketch a shape on a plane, dimension the sketch, and then extrude the sketch into a three dimensional solid. The sketch icon is selected as shown in Figure 5.2 to begin the profile of the bearing bracket.

Figure 5.3 shows tools available to sketch the profile features, including icons for line, circle, or arc. Next, the drawing is dimensioned according to the desired specification using the dimension icon. Once the desired profile has been fully dimensioned, the lines in the sketch will turn from blue to black. After the sketch is fully defined, the profile must be extruded to three dimension using the extrude icon. When extruding, the window will ask for the distance to extrude. Figure 5.4 shows the extruded profile.

The bearing hole must now be cut out of the extruded profile. By selecting a face on the extruded profile, the size of the bearing hole can be drawn as shown in Figure 5.5.

Selecting the extrude cut icon will cut the 1.2 inch diameter hole through the extruded profile. In order to make 0.375 inch holes to mount the bearing bracket, the bottom side of the bracket is selected to sketch upon. The holes are then sketched and dimensioned as shown in Figure 5.6, and the extrude cut icon is selected to cut the 0.375 inch holes through the bottom of the bracket.

The final step in the design is to fillet the sharp corners of the profile. The line to be filleted and then the fillet icon is selected as shown in Figure 5.7. Repeating the procedure will place a fillet on the other side.

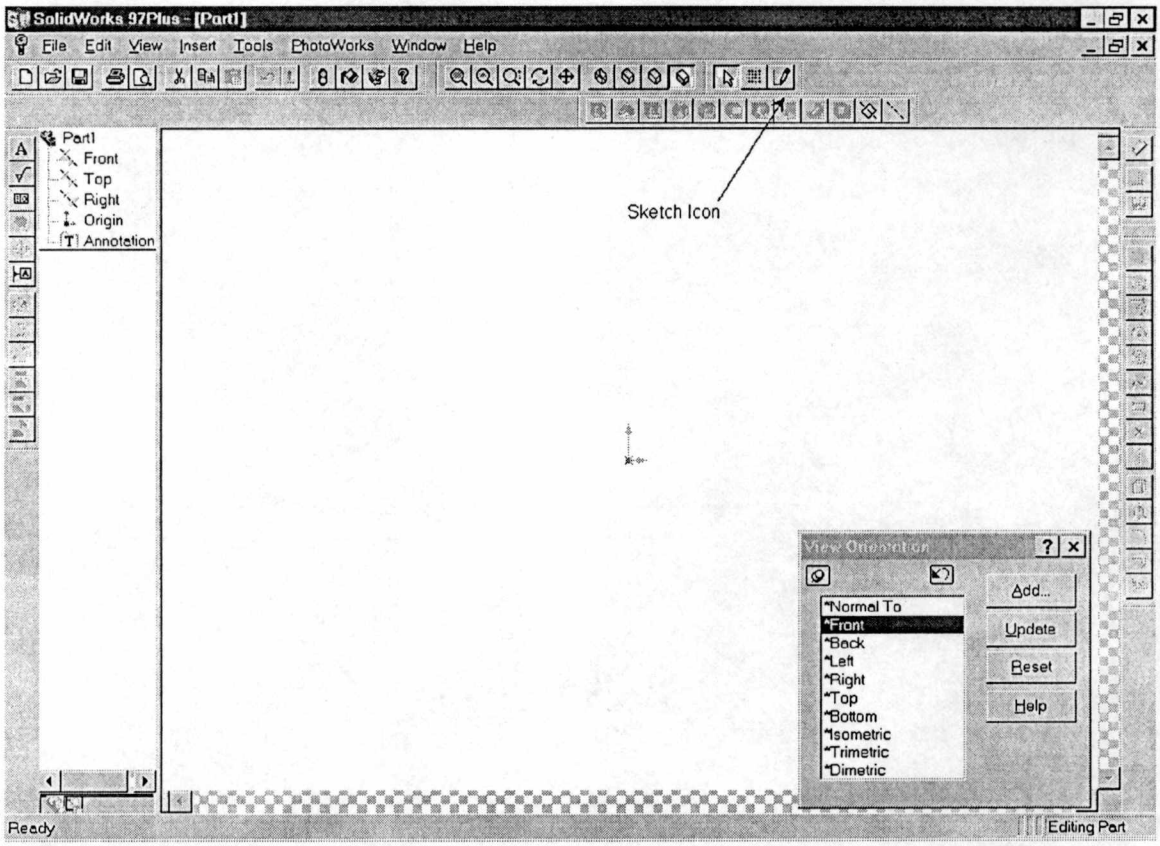


Figure 5.2: SolidWorks97Plus Work Screen

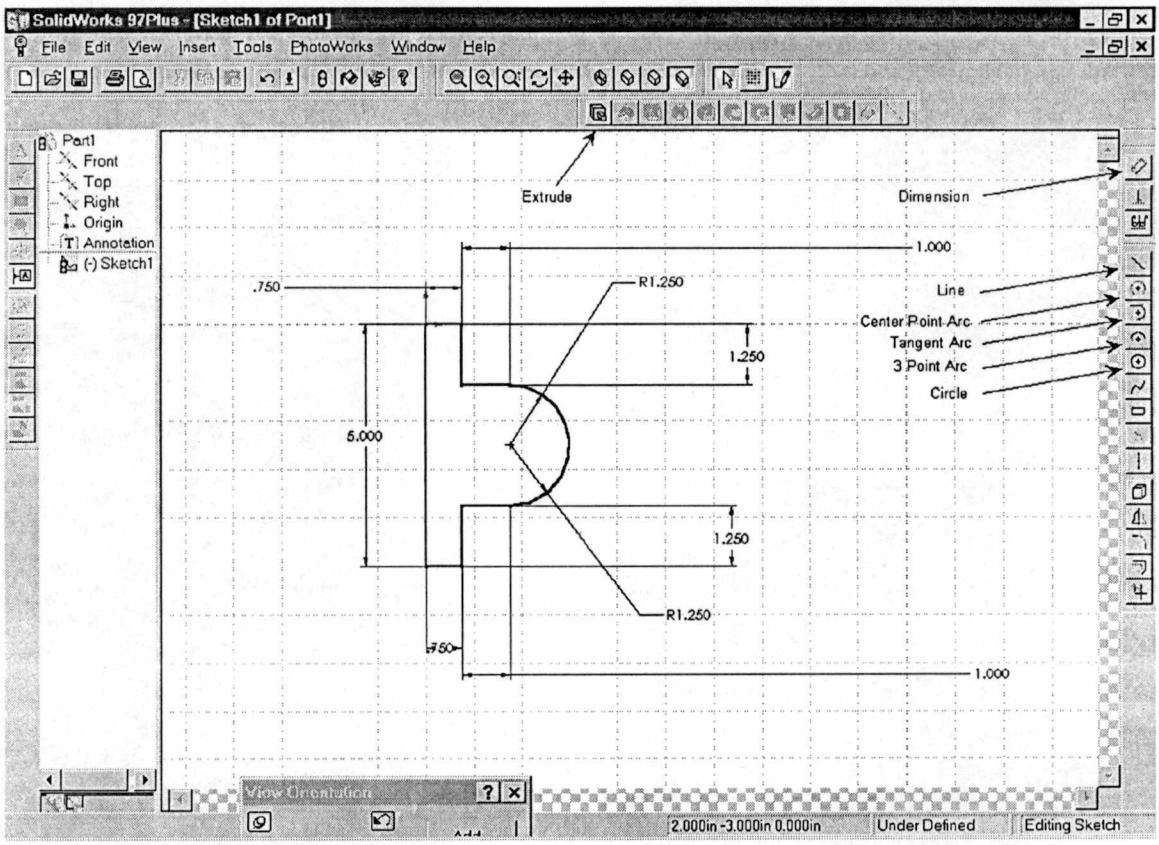


Figure 5.3: SolidWorks97Plus with Profile Sketch

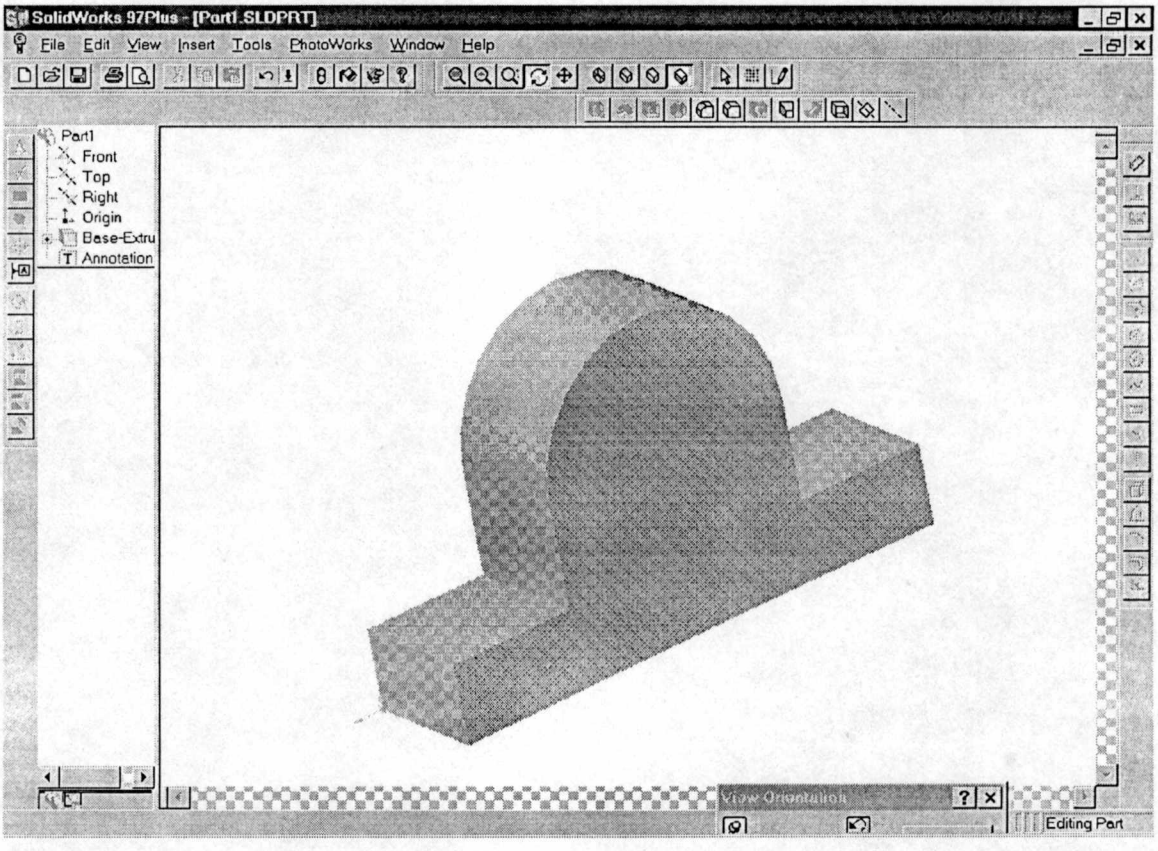


Figure 5.4: SolidWorks97Plus with Extruded Profile

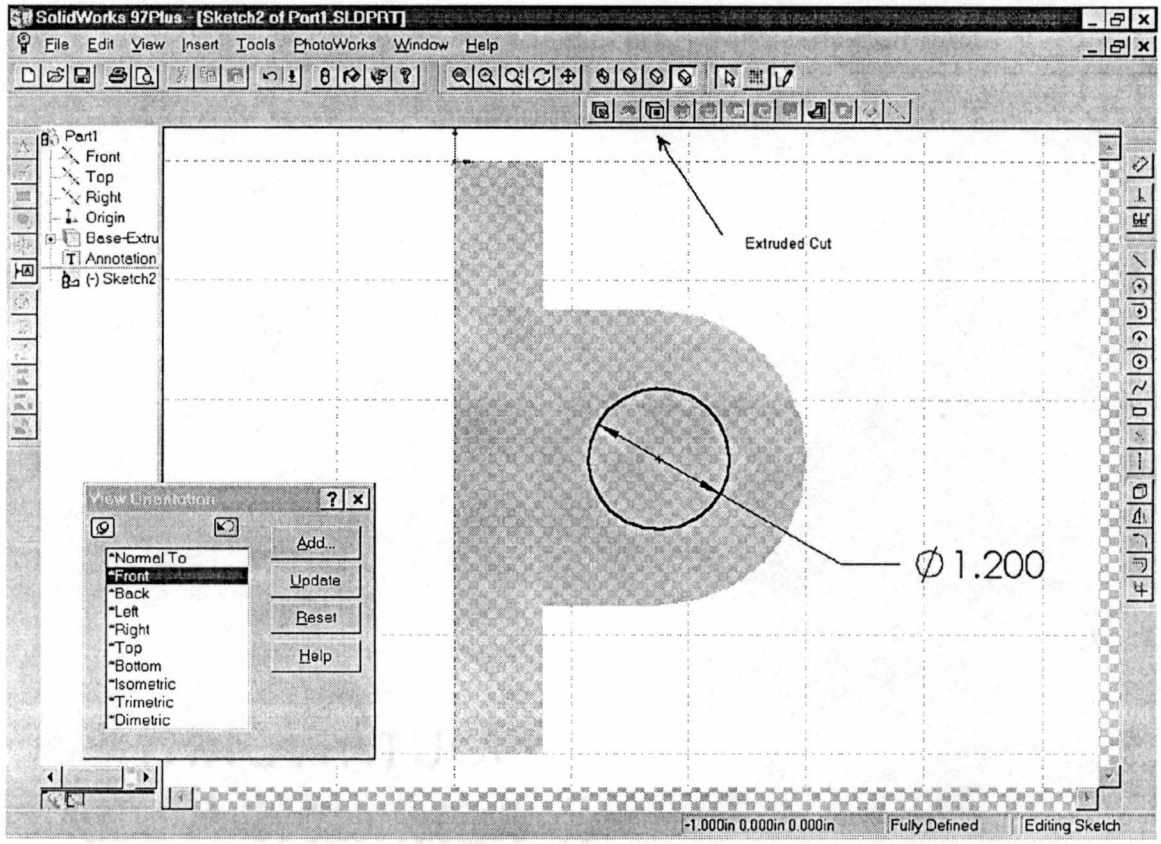


Figure 5.5: SolidWorks97Plus with Bearing Hole Sketch

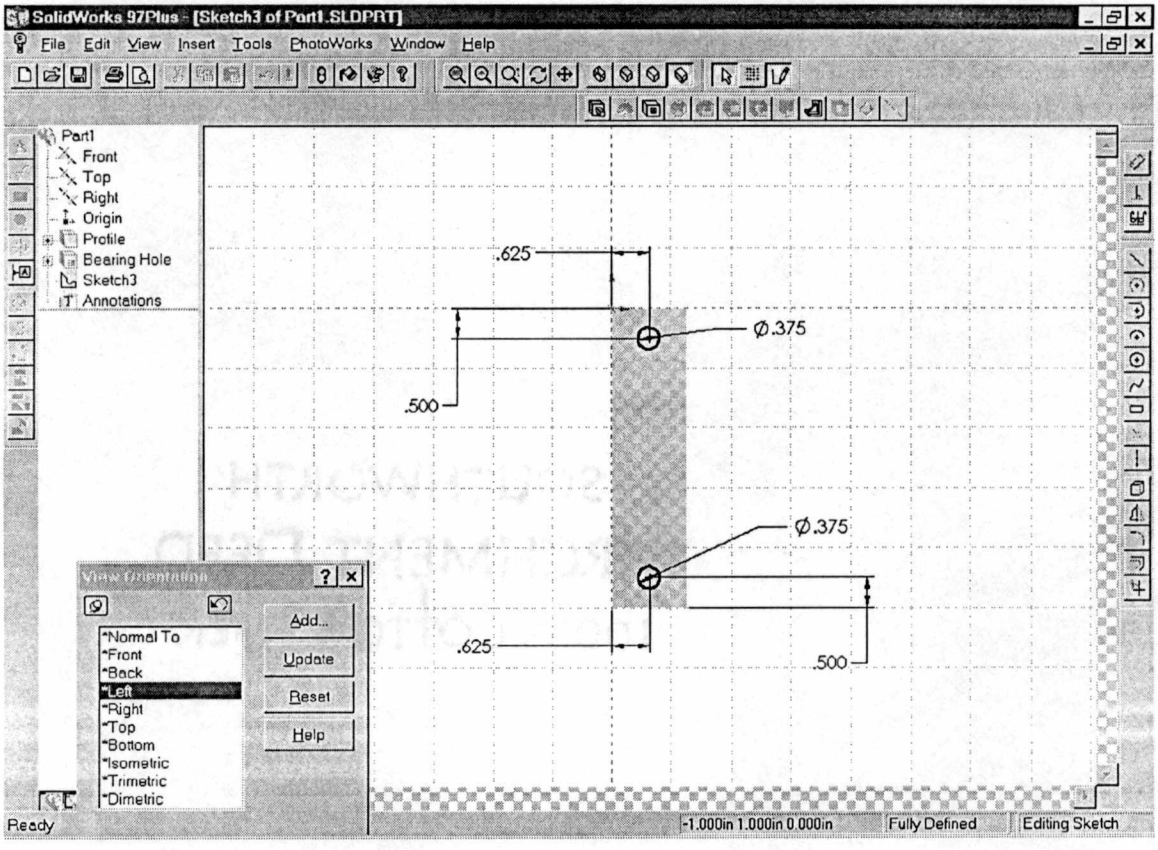


Figure 5.6: SolidWorks97Plus with Bracket Hole Sketch

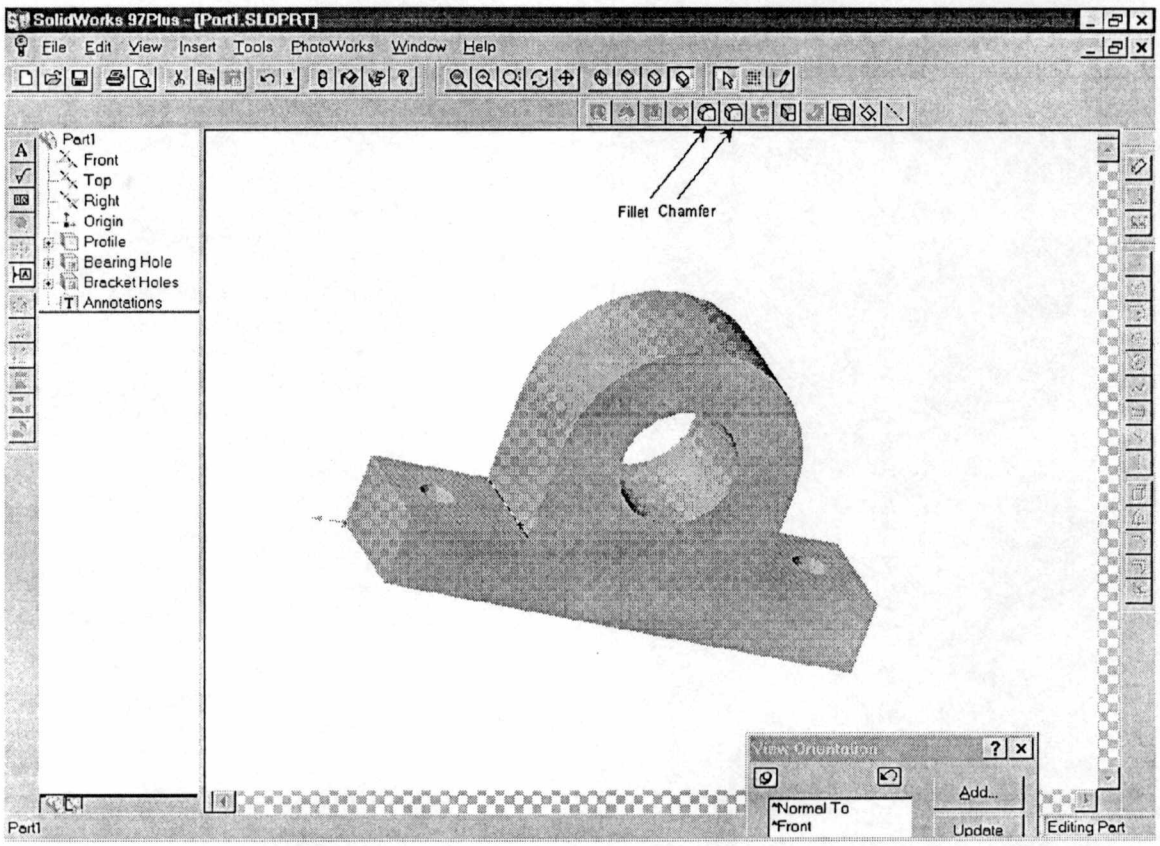


Figure 5.7: SolidWorks97Plus Bearing Bracket without fillets

Figure 5.8 shows the complete bearing bracket model. The feature manager shows the bearing bracket in terms of design features. Profile, Bearing Hole, and Bracket Holes are listed in the feature manager in the order they were created. The sketch associated with this feature can be changed by right mouse clicking the feature in the feature manager and editing the sketch. After the changes are complete, clicking the rebuild icon will rebuild the bearing bracket model with the new changes.

The roll back bar is used to turn back the design process. If a designer wished to place a feature between the Profile and Bearing Hole features, the roll back bar would be dragged until it is between the Profile and Bearing Hole feature. The roll back bar makes the software unaware of the features created below its new position. After the new features are inserted, the roll back bar is dragged to the bottom of the feature manager and the model is rebuilt.

The final step is to click the save icon and save the file to the SURFCAM .DSN file format as shown in Figure 5.9

5.2 SURFCAM 7.0

Figure 5.10 shows the SURFCAM^{TM} operator's window when it is started. On the left side of the window are the menu commands. These commands, similar to pull down menus, provide options after a menu command is selected. Figure 5.10 shows SURFCAM^{TM} at the main menu.

5.2.1 Opening a SolidWorks97Plus file

In order to open a file from SolidWorks97Plus^{TM} select Files, then Open from the menu commands. A window will appear to select the file to open.

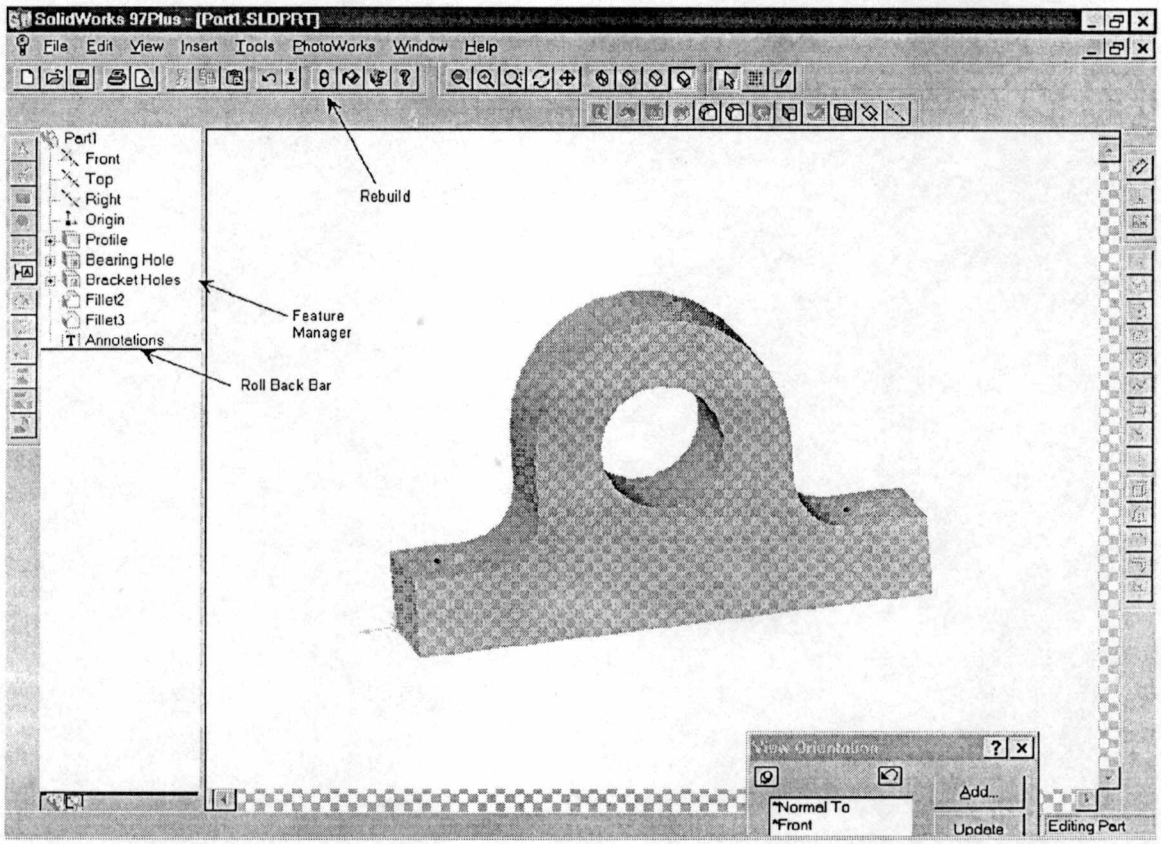


Figure 5.8: SolidWorks97Plus Bearing Bracket Model

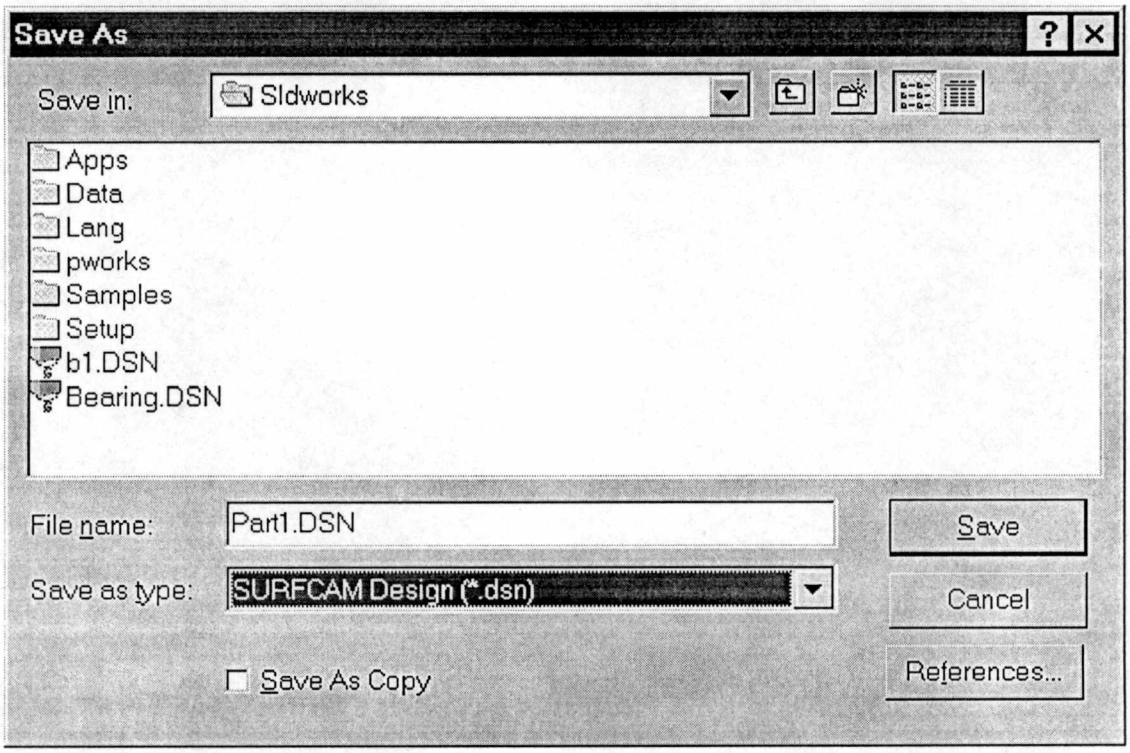


Figure 5.9: SolidWorks97Plus File Saving Window

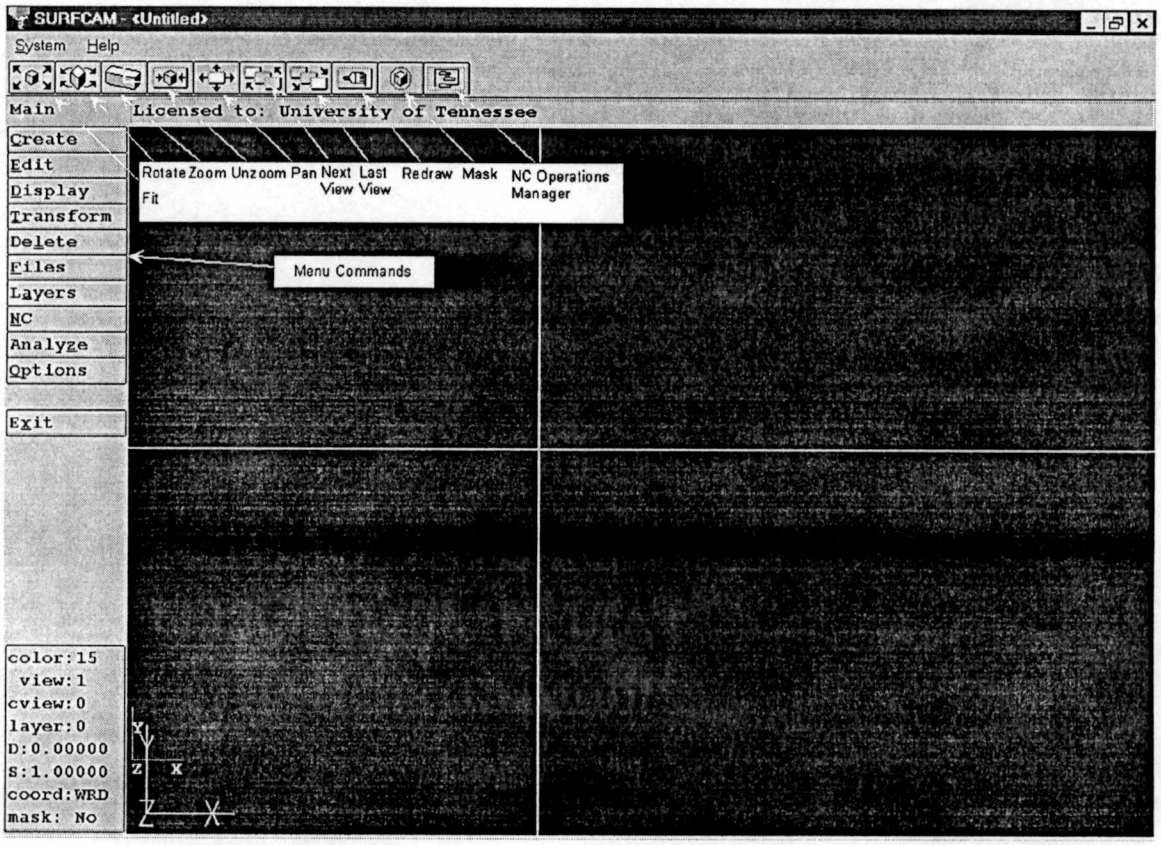


Figure 5.10: SURFCAM Operator's Window

5.2.2 Modifying Surface Arrows

Each surface in SURFCAM^{TM} has an associated arrow as shown in Figure 5.11. These arrows are important in defining the tool path. In order to modify an arrow EDIT and then Surfaces are selected from the main menu. Figure 5.11 shows the menu options available under the EDIT, Surfaces menu commands. Side determines which side of the surface the arrow is located. The side of the surface where the arrow is located determines the side on which the cut will be made. The Corner command determines on which corner of the surface the arrow is located. When a cut is made, the corner on which the arrow is located determines the starting place for the cutter. The Direction command determines which direction the cutter will move as illustrated by the arrow.

5.2.3 Cutting

For this demonstration the 3 axis cut command will be used. From the main menu the following options are selected: NC, NC mode, and 3 axis. Cut from the menu command and the surface to be cut are then selected. Figure 5.12 shows the Tool Information window which is used to select the cutting tool and increment values for the cut. The increment values correspond to how far the tool feeds down on each pass. The Tool Library button is used to select a tool and the window, as shown in Figure 5.13, will allow the designer to select the desired tool. The menu in Figure 5.14 allows the designer to pick the spindle speed, coolant parameters (flood, mist, off), feed rates, and plunge rates. The feed rate is the speed the cutter moves along each pass of the surface. The plunge rate is how fast the cutter is fed deeper into the metal stock for the next pass. The procedure must be repeated to cut the profile of the bearing bracket and bore the hole in the center.

5.2.4 Drilling

For this demonstration the 3 axis drill command will be used. From the main menu command the following options are selected: NC, NC mode, and 3 axis. First, Drill is selected from the menu command. Figure 5.12 shows the Tool Information window which is used to

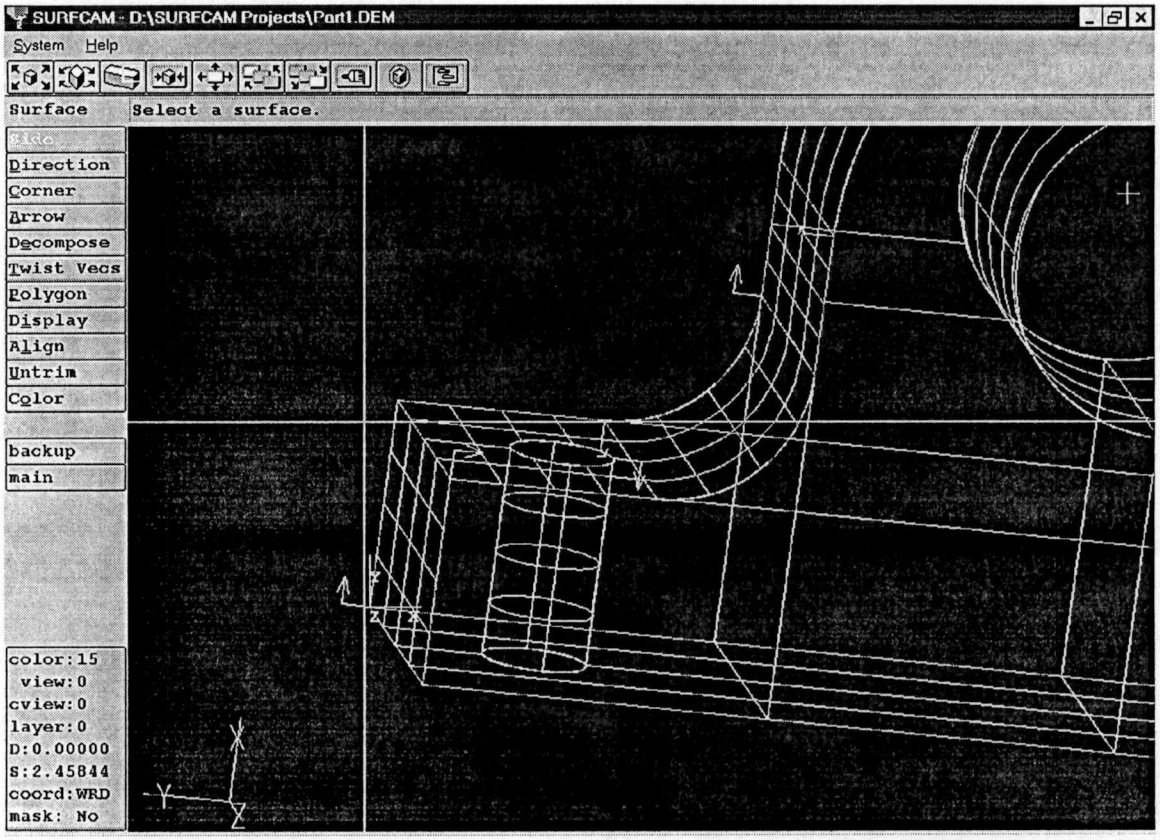


Figure 5.11: SURFCAM Surface Arrows

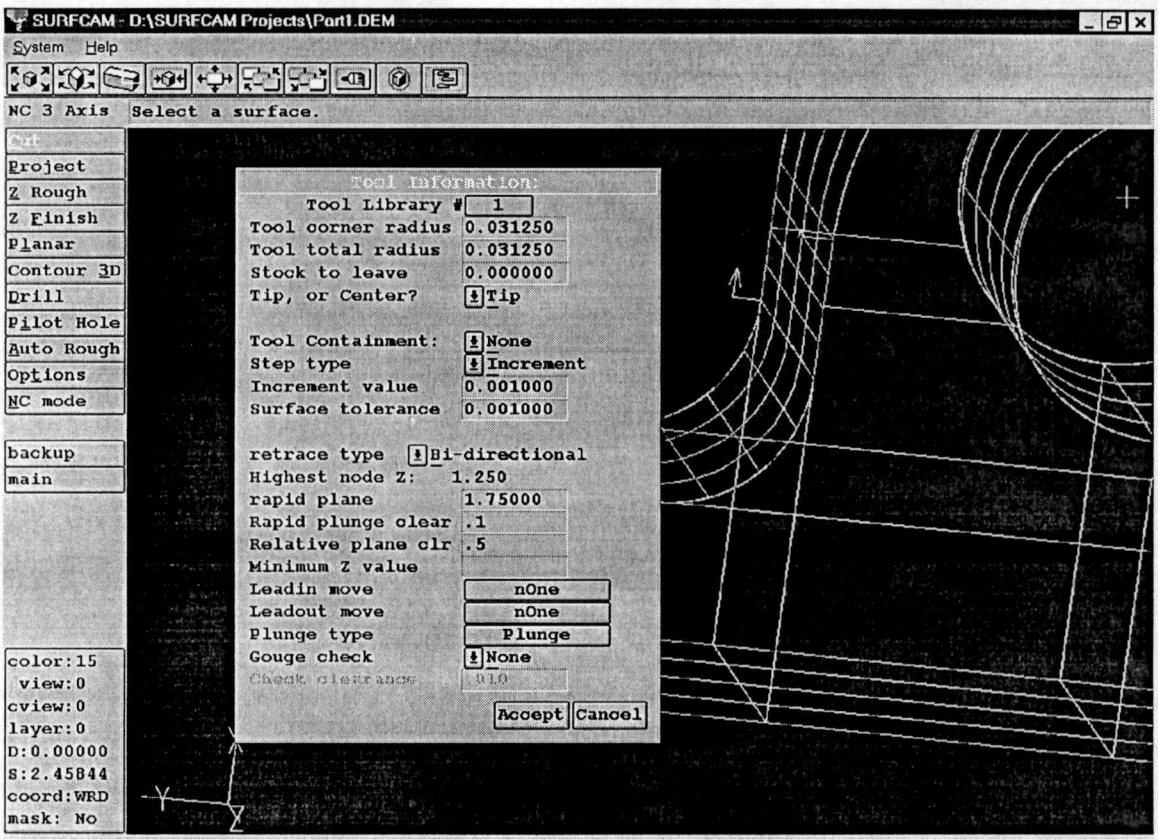


Figure 5.12: Tool Information Window

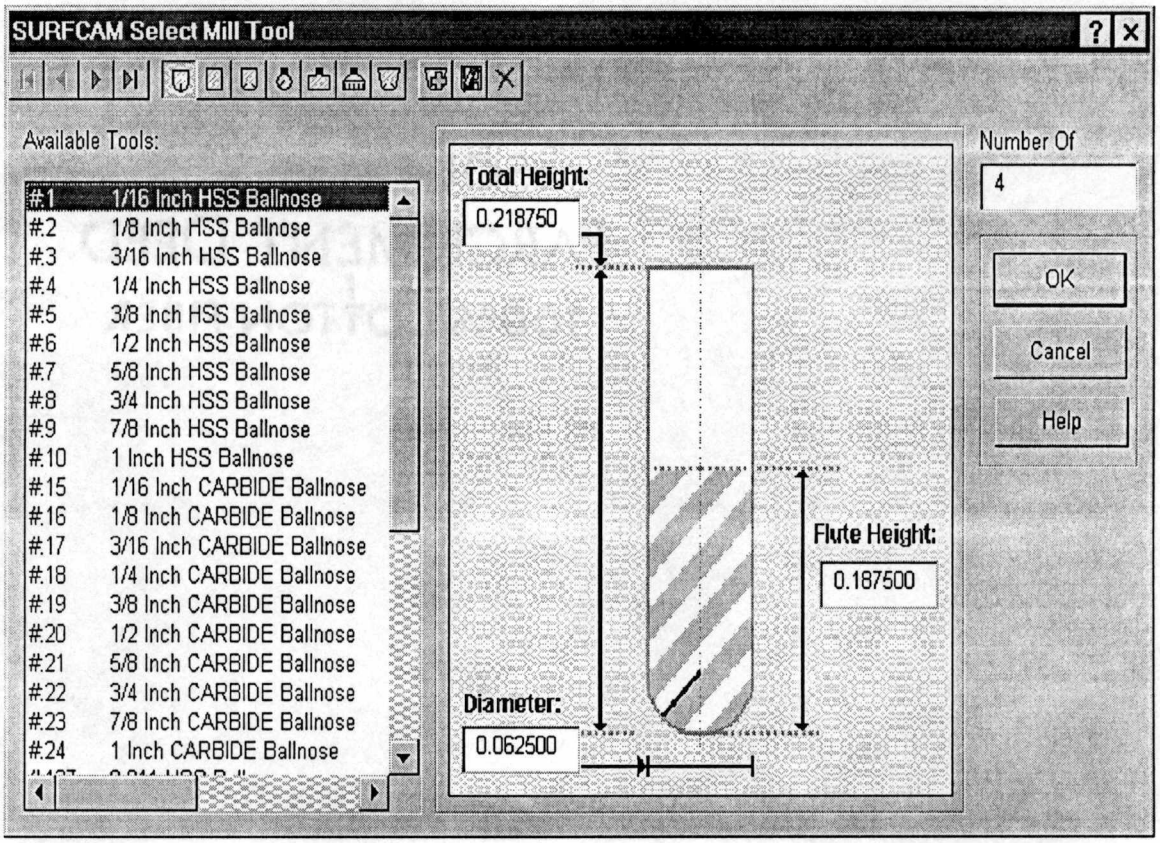


Figure 5.13: Select Tool Window

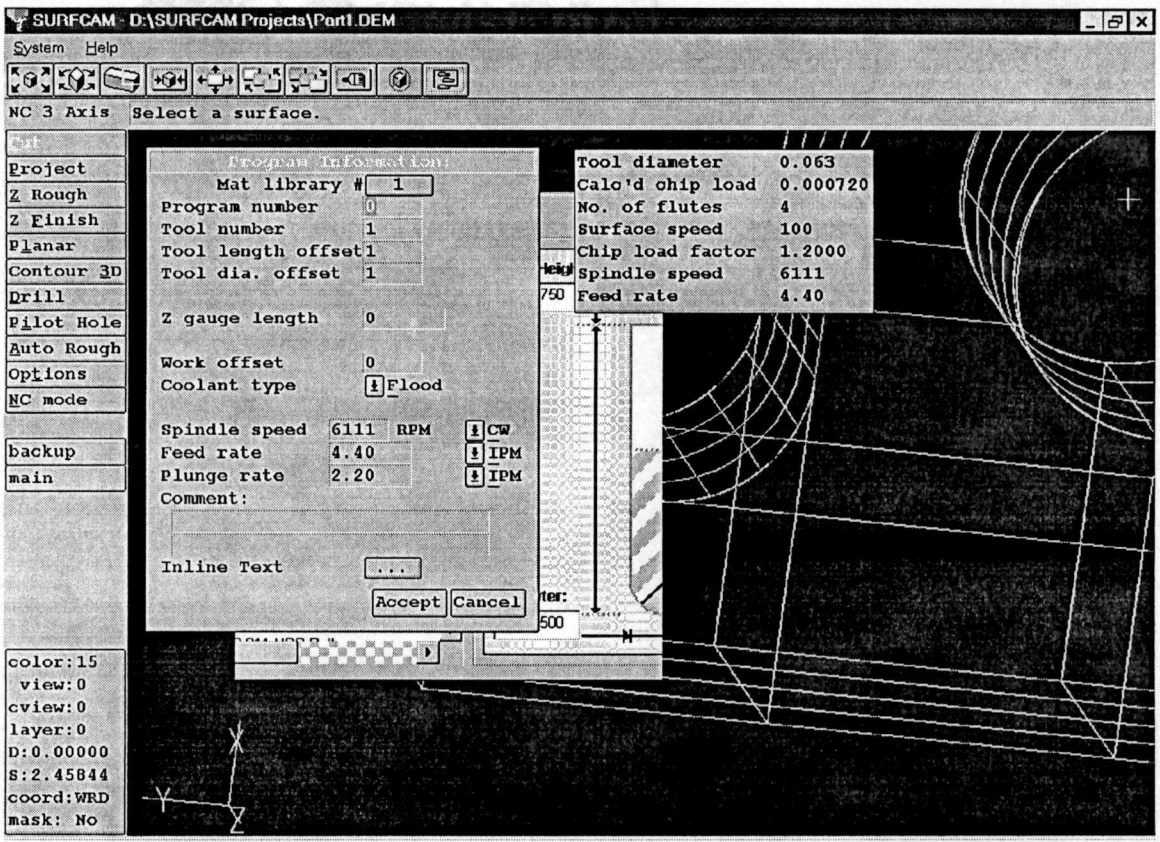


Figure 5.14: Program Information Window

select the drill bit. The Hole Depth determines how deep the hole will be drilled. The Tool Library button, and the SURFCAM Select Drill Tool window shown Figure 5.13 allows the designer to select the desired drill bit size. The Program Information Window, shown in Figure 5.14, allows the designer to pick the spindle speed, coolant parameters (flood, mist, off), and feed rate. The feed rate determines how quickly the drill bit is pushed into the metal. Once all of the drilling parameters have been set, the holes to drill and the Done menu command button are selected.

5.2.5 NC Operations Manager

The NC Operations Manager shown in Figure 5.15 shows the Profile Setup and the Drill Setup. Profile Setup contains all of the cuts to manufacture the profile and bore the bearing hole as shown. The Drill Setup contains the drilling operation to drill the two bracket holes. Selecting the desired setup and pressing the NC Simulate icon will allow the designer to simulate the tool path cutting a piece of stock.

5.2.6 Post Processing

The right hand window in the NC Operations Manager, as shown in Figure 5.15, is a list of several milling machines. Selecting the appropriate machine tool in the window and pressing the Post button will produce the desired part program to run on the machine tool. Since MDSI's OpenCNC^{TM} was not an option, using the Bridgeport Boss 8 will produce the necessary code to run the machine tool using the OpenCNC^{TM} control system. During the conversion to open architecture control, it was found that OpenCNC^{TM} [18] supports most of the Bridgeport machine tool codes [14]. Figure 5.16 shows the machine code produced by the post processor. The first line of the part program shows the G codes used throughout the entire program. Consult [18] to make sure MDSI supports these codes before running the machine tool. If some changes need to be manually performed within the code, the Global Reseq command shown in Figure 5.16 will renumber the blocks of code.

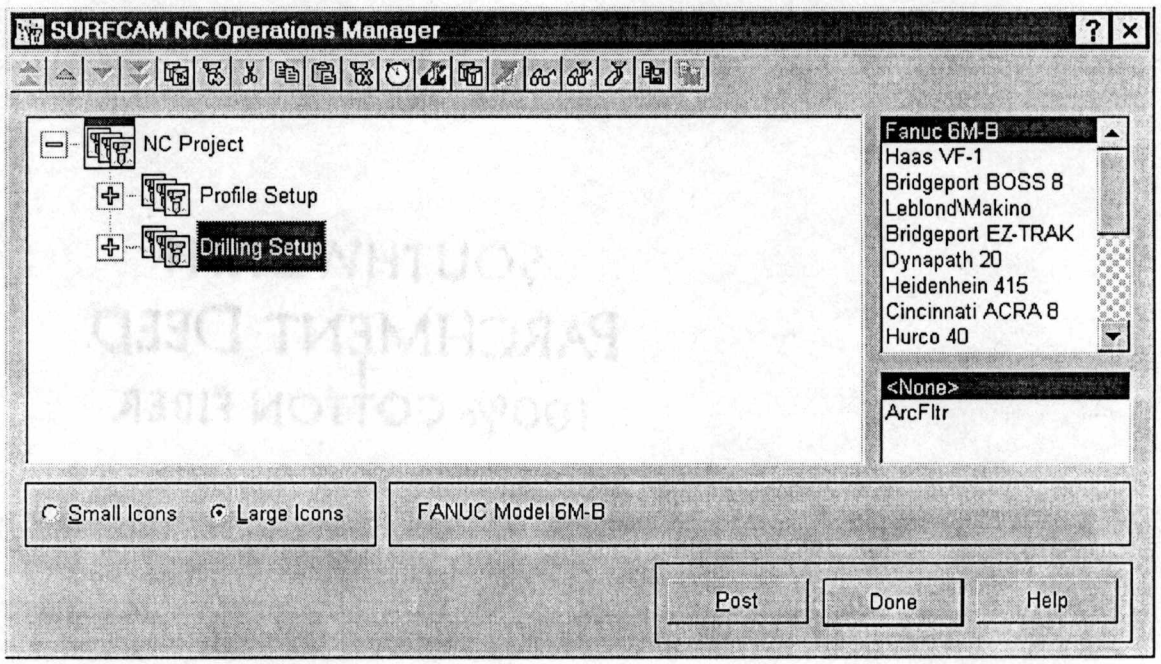


Figure 5.15: NC Operations Manager

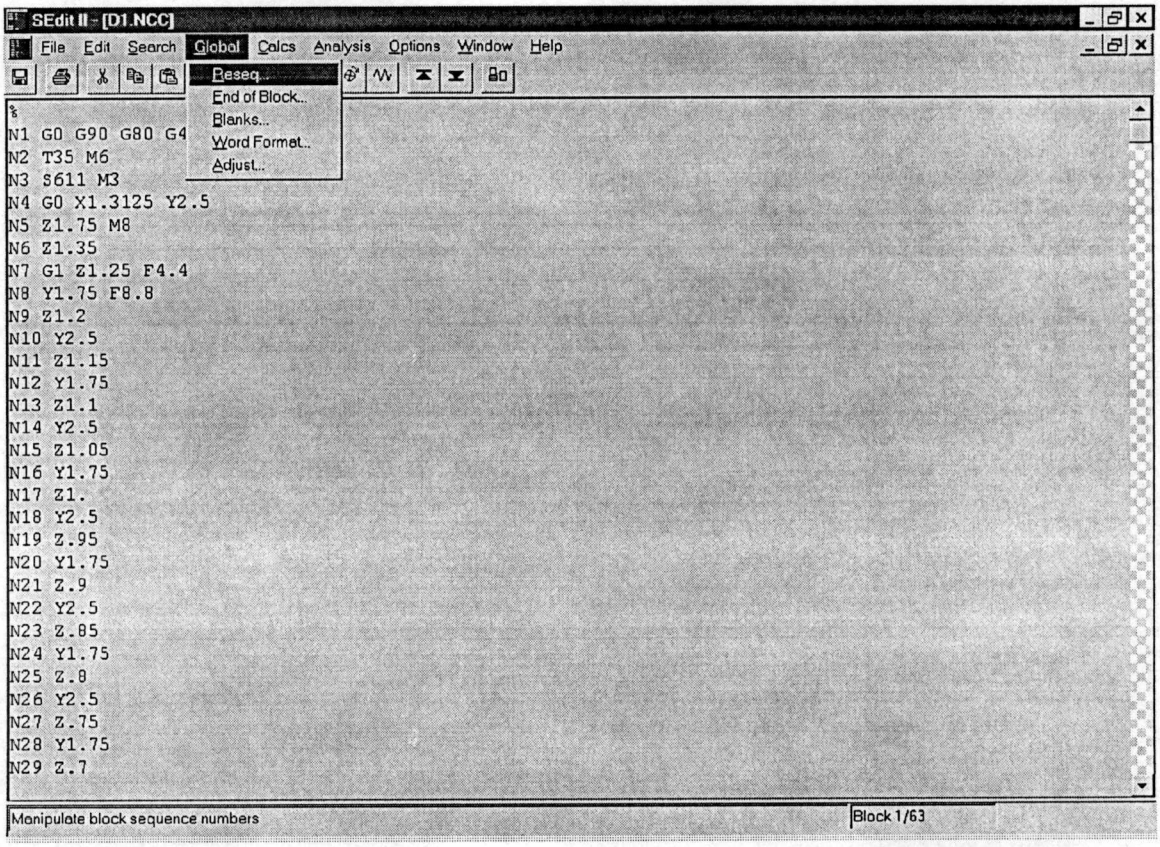


Figure 5.16: Part Programming Editor

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Converting a machine tool to open architecture control is a significant endeavor. A tremendous amount of practical and theoretical knowledge is required in the areas of systems engineering and computer science. If a company desired to convert a machine tool to open architecture control with its own engineering staff, teaming two engineers to work on the problem together would be advantageous. Working in pairs would aid in trouble shooting problems during the machine tool conversion process. Teaming engineers with varied background would be ideal. If one engineer were comfortable with machine tool electrical hardware and another with software, this would allow the members to expand the other's capabilities. Talking through problems with another professional would decrease the overall conversion time.

Open architecture control is the wave of the future in machine tool controls. Machine tool manufacturers will begin to form alliances with producers of open architecture control software. Machine tool manufacturers will offer their customers the option of selecting between different brands of open architecture control software on a new machine tool. It will be interesting to see how the machine tool manufacturers will market their products in the future. In the past machine tool builders could differentiate their product with an extra option on their proprietary control system. With multiple manufacturers selling the same control system software, it appears the only deciding factor between products will be

cost. Likewise, retrofitting a previously used machine tool with new controls has become economically attractive. The retrofitted machine tool market will significantly cut into new machine tool sales.

Open architecture control will cause faster progress in machine tool control innovation. Software providers will constantly be competing with their competitors to produce a better product. Research will become an integral part to improving the existing software. These innovations will be quickly passed along to the customer in the form of software upgrades. Technical documentation will be a significant factor in the success an open architecture control software package. The conversion manuals must be written with enough generality to understand the overall process but with sufficient detail to make the conversion process as simple as possible. The innovations cannot be passed on to the customer effectively, if the software provider's engineering staff has to install all of the software upgrades. Clear documentation and instructions for conversions and upgrades will be essential.

Low-cost PC based Computer aided drafting (CAD) and computer aided manufacturing (CAM) technology is available today. CAD and CAM packages are robust enough to handle most situations. Because the software is robust, using the software is not a trivial task. To fully use the capabilities of the package, an engineer needs to attend the formal training. Learning all of the capabilities from the manuals is not practical. CAD and CAM package must continue to seek to make their software, documentation, and instructional materials more intuitive.

6.2 Future Work

Several useful projects could be performed on the Bridgeport milling machine in the future. The Bridgeport mill has not yet been thoroughly tested for accuracy and repeatability. A case study of how well MDSI's OpenCNC software can position the axes would be ideal. The OpenCNC software could subsequently be enhanced to improve the axis motion.

The open architecture controller could be scaled to monitor process parameters on the milling machine. Milling machine parameters such as tool wear and contact force could be

monitored by adding additional hardware and integrating it into the OpenCNC software. Additional algorithms could be added to the software to both account for and control the parameter variations in the milling process.

Communication between the server and client could be increased to provide information on the performance of the milling machine. The client could report on the number of completed part programs and the current state of operation for the milling machine. This type of reporting could allow operators and engineers to monitor the milling machine's performance and make informed decisions regarding maintenance and scheduling.

Finally, an expert system could be added to the OpenCNC software to teach an operator how to start and run the machine tool. This would be an online tutorial to teach all of the machine tool functions and operations ranging from enabling the machine tool to physically producing a part.

Bibliography

Bibliography

[1] *SolidWorks97Plus User's Guide.*

[2] *SURFCAM Version 7.0 User's Manual.*

[3] *Center For Industrial Services: Sometimes it takes more than a loud knock to open the right doors. Sometimes you need a key.*

[4] J. Francis Reintjes. *NUMERICAL CONTROL Making a New Technology.* Oxford University Press, 1991.

[5] GM Powertrain. Requirements of open, modular architecture controllers for applications in the automotive industry. Version 1.1, 1994.

[6] GM Powertrain. Open, modular architecture controls at gm powertrain: Technology and implementation. Version 1.0, 1996.

[7] Mark Albert. Open-architecture cnc closes servo loop in software. *Modern Machine Shop*, July 1997.

[8] John Teresko. Opening up oppurtunities. *Industry Week*, April 1997.

[9] Su-Chen Jonathon Lin. *COMPUTER NUMERICAL CONTROL: Essentials in Programming and Networking.* Delmar Publishers, 1994.

[10] Giorgio Rizzoni. *Principles and Applications of Electrical Engineering.* Irwin Publishers, 1993.

[11] *ISaGRAF Version 3.00 User's Guide.*

[12] *Model 12 Encoder Interface Industry Pack User's Manual.*

[13] *OpenCNC Hardware Installation Manual.*

[14] *BRIDGEPORT SERIES 1 V2XT Installation and Maintenance Manual Machines 269 and Subsequent.*

[15] *OpenCNC Software Installation Manual.*

[16] *GreenSpring Computers:User Manual IP-DAC.*

[17] *OpenCNC Variable Dictionary.*

[18] *OpenCNC Mill Programming Manual.*

[19] *QNX Operating System: Utilities - A to M.*

[20] *QNX Operating System: Utilities - N to Z.*

Appendices

Appendix A

Operator's Manual

A.1 Control Panel Layouts

Figure A.1 shows the Bridgeport milling machine with the OpenCNC^{TM} control software and hardware installed.

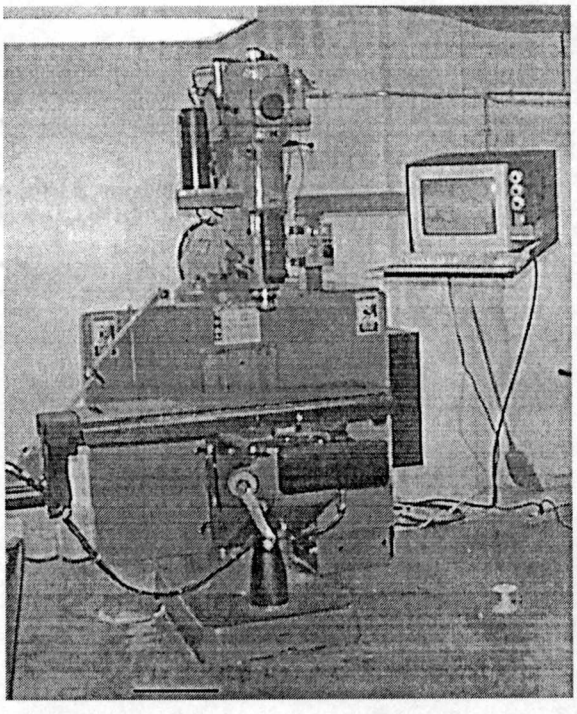


Figure A.1: Bridgeport Milling Machine with OpenCNC^{TM} control

A.1.1 Miscellaneous Hardware

Figure A.2 shows the way lube container, the coolant container, the air intake, and the solenoids. The way lube must be full in order to properly lubricate the axis lead screws. The coolant container has manual knobs to adjust the amount of mist coolant being sprayed on a part during manufacturing.

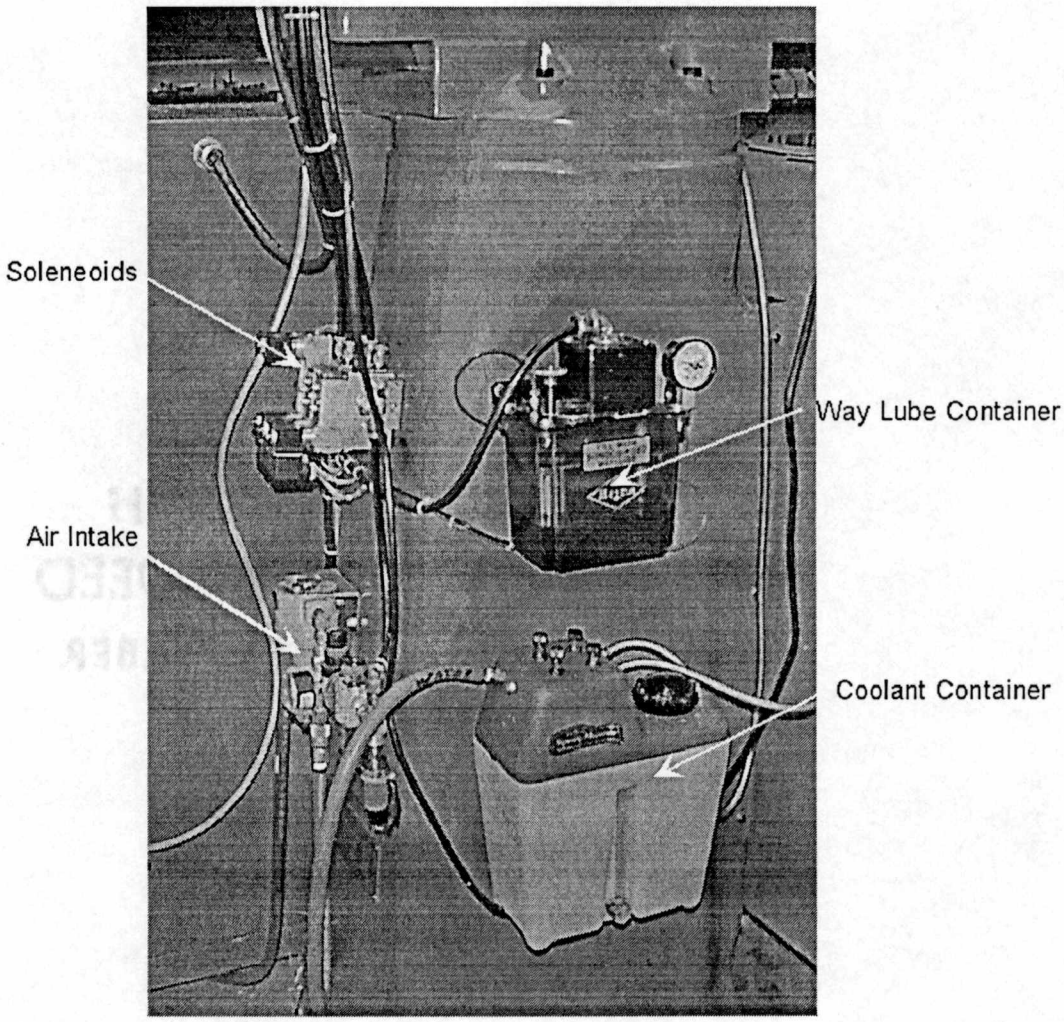


Figure A.2: Miscellaneous Hardware

A.1.2 Hardwired Switches

Figures A.3 and A.4 shows the hardwire switches available on the Bridgeport milling machine. ESTOP1 and ESTOP2 are the emergency stop switches. These switches will stop all axis and spindle movement in the event of an emergency. The CYCLE_START push button starts the execution of a part program. The X, Y, and Z overtravel bypass switches are used to bypass the overtravel switches in the emergency stop circuit. These overtravel bypass switches will allow an operator to move the axes off an overtravel switch. Be careful and jog the axes slowly when moving off of an overtravel switch. Moving the axis in the wrong direction could cause serious damage to the axis drive.

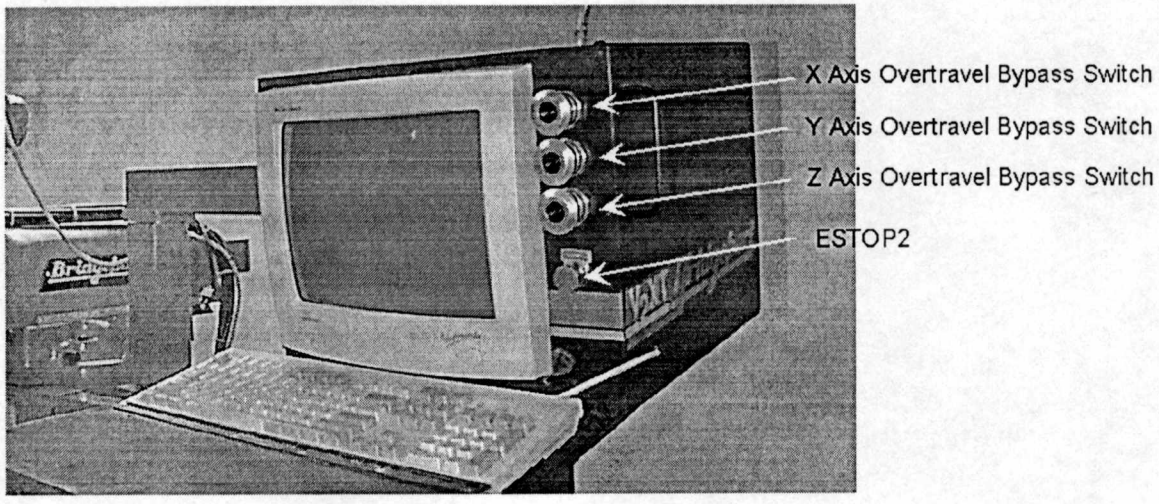


Figure A.3: Hardwire Switches beside the touch screen

A.1.3 Touch Screen Layout

Figure A.5 shows the touch screen after the operator has completed the procedure in Subsection A.2.1. The MDSI Operator Panel controls most of the functions on the machine tool and is shown in Figure A.6. Spindle control button allows the operator to manually set the spindle speed. The Coolant Flood and Coolant Mist Buttons are not currently in use. Coolant mist is manually turned on at the coolant container as shown in Figure A.2. The Bridgeport milling machine does not have a coolant flood capability. The Rpm%, Fpm%,

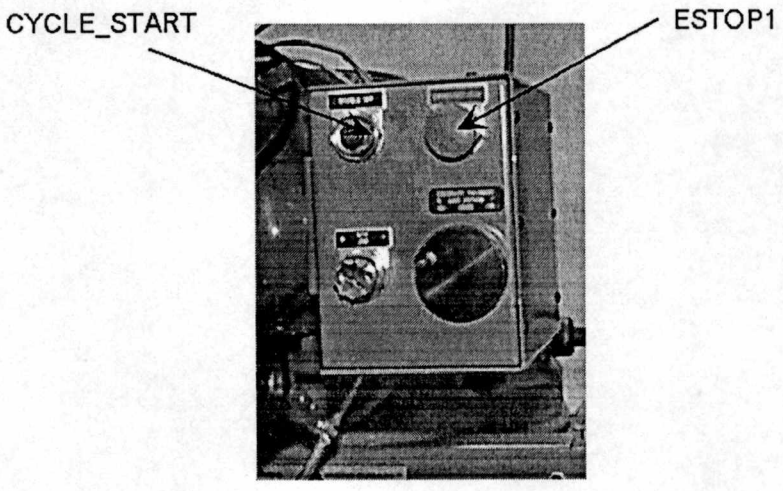


Figure A.4: Hardwire Switches

Rap%, and Jog% buttons are the spindle speed, feed rate, rapid movement, and jog rate override slider bars. These % override buttons allow the operator to override the spindle speed, feed rate, rapid movement, and jog rates specified in a part program. The OpStop button allows for optional stops while running a part program. The Mode button chooses whether the machine is in jog or run mode. In jog mode the operator manually moves the axes with jog button. In run mode the machine tool will begin executing a part program. Axes Jog button allows the operator to manually jog the axes of the milling machine. The Startup button has three options: Sync, Machine Start, and Machine Stop. Machine Start option energizes the emergency stop circuit. The Machine Stop button deenergizes the emergency stop circuit. The Sync option syncs or homes the machine to define an inertial or global reference frame.

The MDSI 1 Position window as shown in Figure A.5 shows the current position of the milling machine and the distance to travel. The MDSI PROGRAM window displays the part program to be executed as shown in Figure A.5. The MDSI PROGRAM window allows the operator to view and edit the part program. The MDSI button in the top right hand corner of Figure A.5 contains options available to operator such Fixture Offset Display and Tool Offset Display.

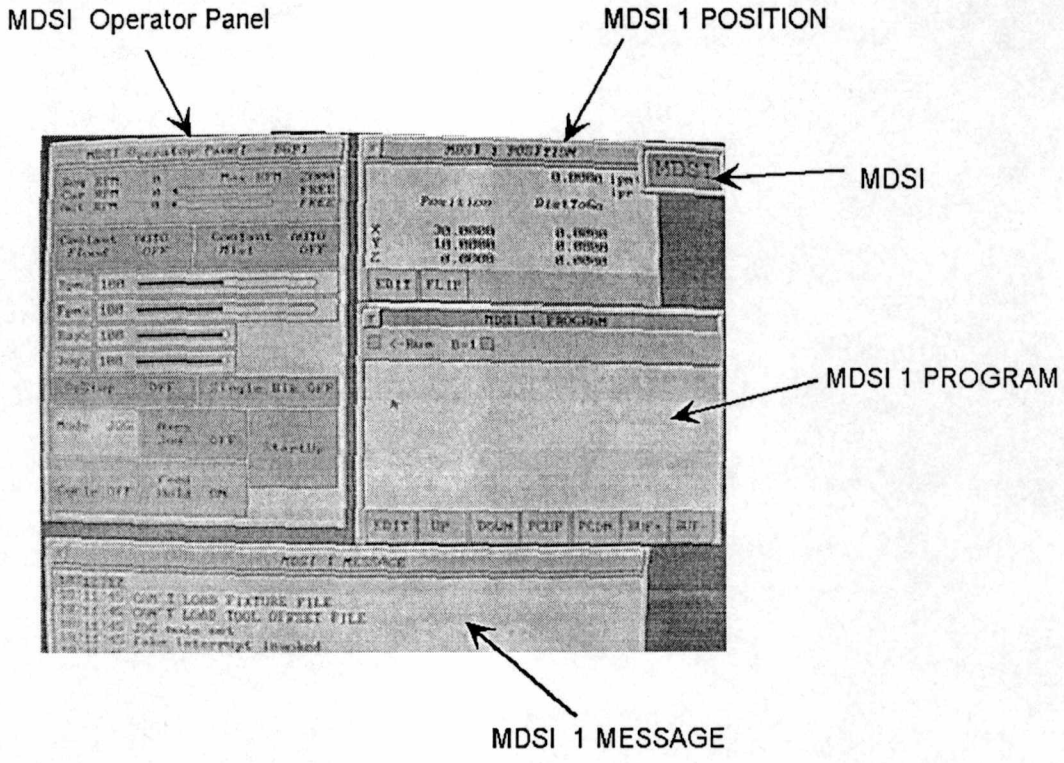


Figure A.5: OpenCNC Window

A.2 Startup Procedure

The following outlines the procedure for operating the milling machine:

1. Turn on the Milling Machine
2. Energize the Emergency Stop Circuit
3. Sync the machine
4. Load the part program
5. Determine Fixture Offsets
6. Determining Tool Offsets
7. Starting a Part Program

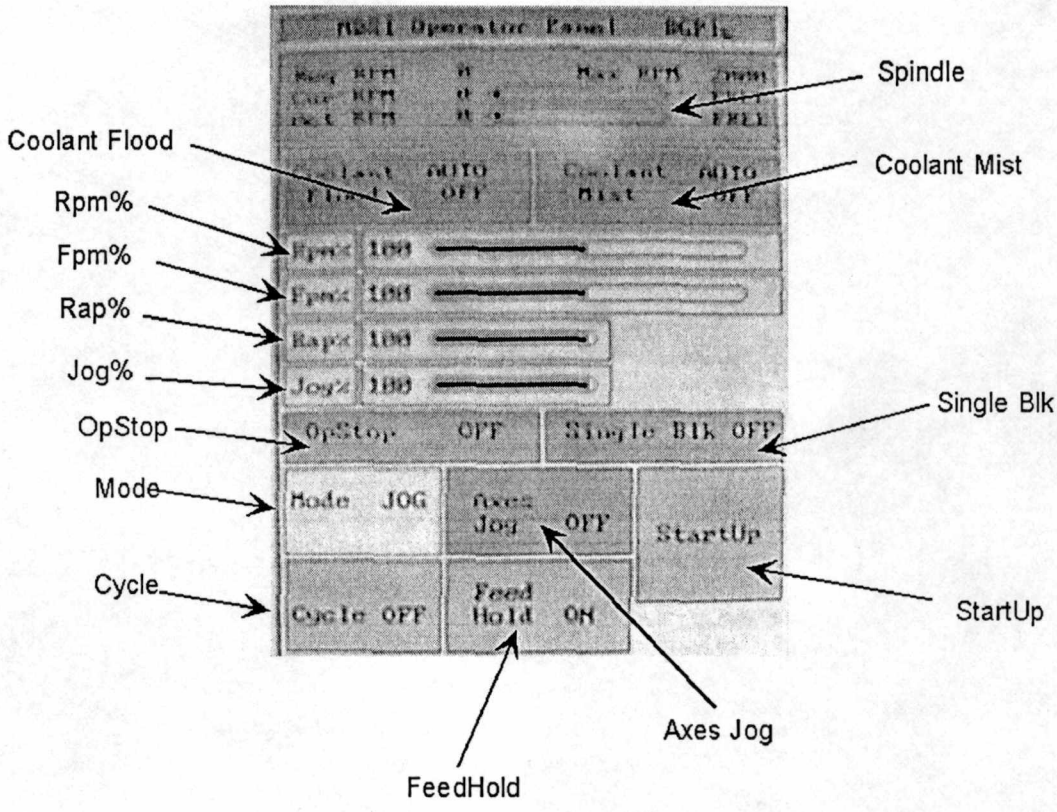


Figure A.6: MDSI Operator Panel - BGP1 Window

A.2.1 Turning on the Milling Machine

Turn on the computer and the touch screen monitor only. DO NOT TURN ON THE MILLING MACHINE UNTIL YOU HAVE LOGGED IN!!!!!!! If the computer has been turned on correctly, the screen will prompt you with

login:

Type "bridge" and press the enter key. When the computer asks for a password, type "bridge" and press enter. If you have logged in correctly, the prompt will read:

{bridge}:

Type "runbridge" and press enter to start OpenCNC. If OpenCNC has started correctly, Figure A.5 will appear on the touch screen. Now the milling machines power can be turned on by turning the machine power switch shown in Figure A.7. NEVER TURN OFF THE

COMPUTER WHILE THE MILLING MACHINES' POWER IS STILL ON !!!!!!!!!

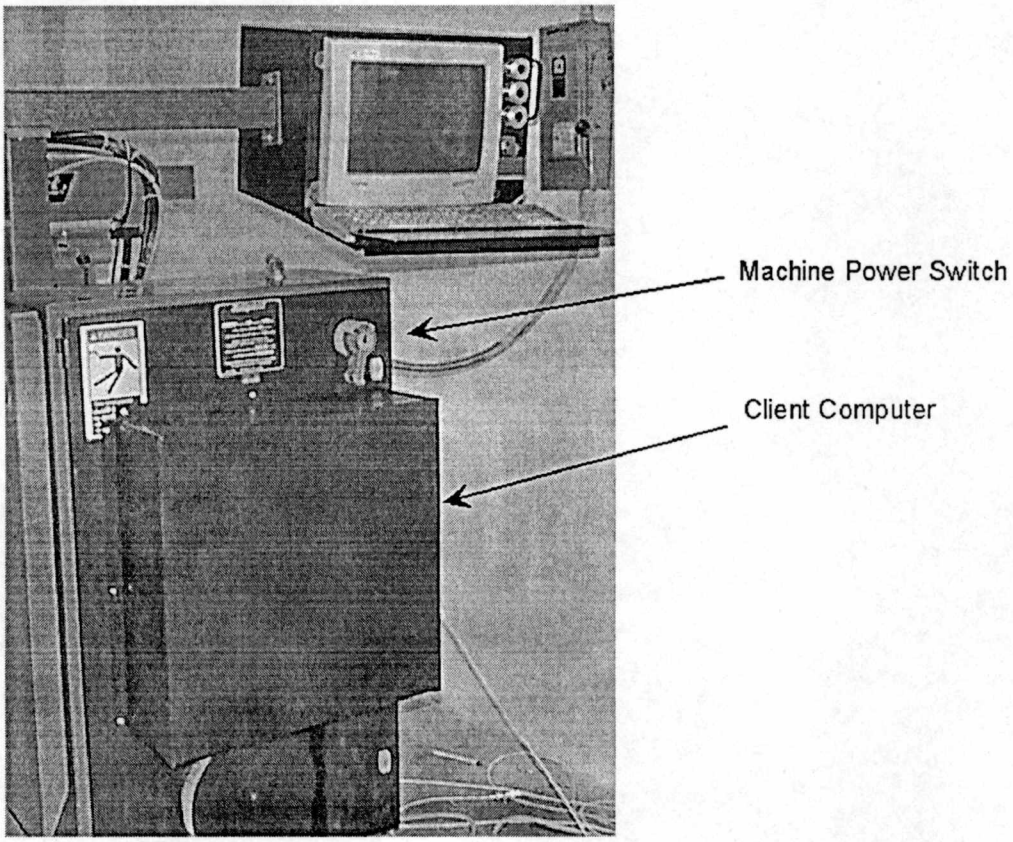


Figure A.7: Machine Power Switch and Computer

A.2.2 Energizing the Emergency Stop Circuit

When the operator has successfully completed the instructions in Section A.2.1, the next step is to energize the emergency stop circuit. Press the StartUp button and then press Machine Start. If the emergency stop circuit becomes energized, the MDSI 1 MESSAGE will read, "machine enabled."

A.2.3 Syncing the Machine

After the operator has successfully energized the emergency stop circuit outlined in Section A.2.2, the next step is to sync or home the machine. Figure A.8 shows the milling machine

with all of the axes in the home position. The X and Y axis home positions are located on the negative end of travel. The Z axis home position is on the positive end of travel. Press the Axes Jog button and select Slide option. Select the axis and slide the axis to the home position. The axis will stop moving once it has reached the home position. The MDSI 1 MESSAGE will read, "X REF." when the x axis is on the reference or home position. After all the axes are in the home position, press the StartUp button and the Sync option. Once the machine has successfully synced, the MDSI 1 MESSAGE window will display, "sync done."

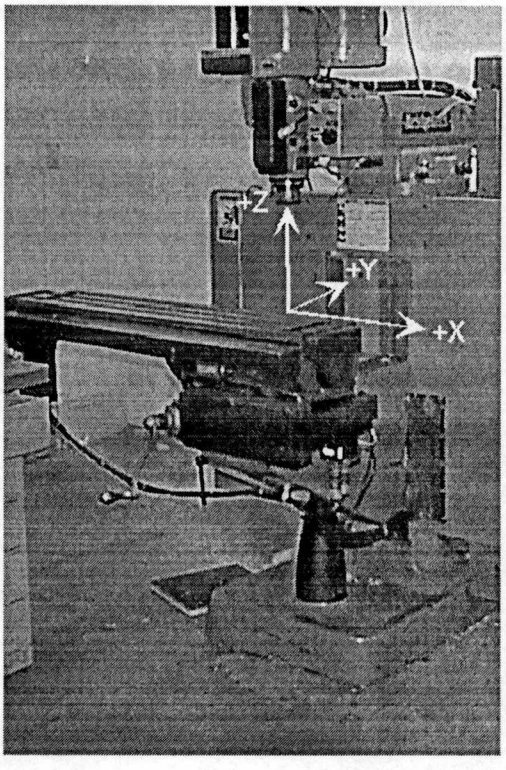


Figure A.8: Home Position

A.2.4 Loading a part program

After the machine tool has been synced as outlined in Section A.2.3, the next step is to load the part program to be executed. Press the edit button in the MDSI 1 Program window as shown in Figure A.9. Select the Load option and type the path to the desired part program.

The default path used by OpenCNC is

/home/bridge/programs/filename

Assuming the machine tool was booted from the client hard disk and not over the network, a program can be loaded from a floppy disk with the following path

/dos/a/filename

A file can be loaded from the Windows 95 partition with the following path

/dos/c/filename

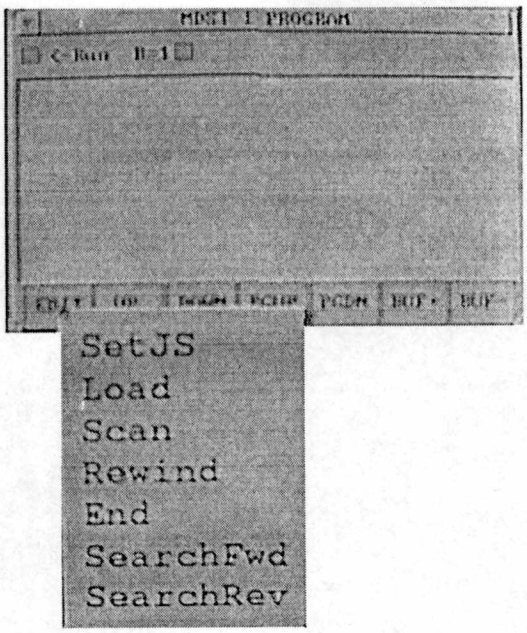


Figure A.9: MDSI 1 Program Window

A.2.5 Determining Fixture Offsets

All part programs are referenced according to some coordinate system attached to the part. Machinists typically attach their coordinate system to the corner of the part to be manufactured. Therefore, the coordinates of the corner relative to the global coordinates

can be determined by "touching up" on the part. "Touching up" refers to a machinist manually moving the tool to physically touch the three sides of a rectangular piece of stock. By touching the part on three sides, the coordinates of the corner can be determined. These values need to be entered into the Fixture Offset Display window. Press the MDSI button in the top right hand corner of the touch screen and then select the Fixture Offset Display option. Part programs use E# commands to note offsets between the global coordinates and the work piece. # is the offset number used in the part program. Enter the values from the touching up procedure in the Fixture Offset Display window in the # row and then close the window. The Fixture Offset Display window must be closed in order for OpenCNC^{TM} to note the changes to the Fixture Offset Display. See [18] for more information on Fixture Offsets.

A.2.6 Determining Tool Offsets

During the touch up procedure the corner of the part was not entirely determined. There was an offset due to the cutting radius of the tool used to "touch up." The offset due to the tooling radius must be entered into Tool Offset Display window. Part programs use T# commands to determine what size tool is being used. # refers to the tool number used in the part program. Enter the tool offset values in the Tool Offset Display window in the # row and then close the window. The Tool Offset Display window must be closed in order for OpenCNC^{TM} to note the changes to Tool Offset Display. See [18] for more information on Tool Offsets.

A.2.7 Starting a Part Program

"Dry running" a part program is generally a good machining practice. "Dry running" refers to running a part program without a tool or part to see if the desired path, feed rates, and spindle speeds are correct. Even if the program has been dry runned, it is a good idea to execute the first several blocks of a part program block by block before letting the machine run at full speed.

First, turn the Single Blk button to ON. This will allow the program to only run one block at a time. Press the Mode button and then press Run. To execute the first line of the program press the Cycle start button. Repeat the process to execute the next block of the part program. Once the operator is sure the program is executing properly, turn the Single Blk Off. This will allow the program to execute each block of code sequentially without the operators permission. NOTE: DO NOT LEAVE THE MACHINE TOOL UNATTENDED!!!!!!! Once the part program has been fully executed, select Mode and Jog to setup the machine to execute the next part program.

Appendix B

Accounts and Passwords

Two accounts were setup on the server and the client. Root is the highest level adminstrative account or super user. bridge is the Bridgeport V2XT machine account to be used by operators when using the machine tool. bridge's priveledges are restricted while root is unrestricted. Table B.1 shows the accounts and their passwords.

To change the authorization level within the MDSI pull down menu, select authorization and type the following

will

The password "will" gives the operator the highest level of authorization which allows access to all the MDSI functions such as: winDIO, winVar, and winFeShow.

Account	Password
root	barker11
bridge	bridge

Table B.1: Accounts and Passwords

Appendix C

QNX Information

C.1 Common Commands

Some commonly used QNX commands are listed in Table C.1. For more information on QNX commands consult [19] and [20].

Command	Description
pwd	Lists the present working directory.
cd	Change directory command.
ls	List the contents of the present working directory.
ls -l	Lists the contents of the present working directory with read, write, and execute priveledges.
windows	Starts QNX windows.
vedit	Starts the vedit text editor.
sin	Lists the jobs currently being executed by QNX.
slay sinname	Slay will kill the sinname job being executed by QNX.
mkdir directoryname	Creates a new directory.
rmdir -R directoryname	Removes a directory and all its contents.
cp filename filenamenew	Copies a file from one name to another.
cd	Change directory command.

Table C.1: Common QNX Commands

C.2 Formatting a floppy disk for the QNX Operating System

To move QNX file from one computer to naother using a floppy disk is desirable. Although files could be transfered using a DOS disk, the DOS format sometimes corrupts a QNX file. To format a floppy disk to the QNX format, insert the floppy disk and type the following

at the QNX command line:
dinit /dev/fd0

C.3 Mounting & unmounting a QNX floppy disk

In order for the QNX operating system to recognize a floppy disk, the disk must be mounted. To mount a QNX disk, type the following at the QNX command line:
mount /dev/fd0 /mf
This command will mount the contents of the floppy drive to the /mf directory.

Once the disk has been used, the disk must be unmounted. To unmount a floppy disk type the following at the command line:
umount /mf
Note: a floppy disk must be unmounted before a new disk can be mounted. DO NOT REMOVE A FLOPPY DISK WITHOUT UNMOUNTING THE DISK FIRST!

Appendix D

sysinit.1

```
#
# Example sysinit for a server machine
#

echo
echo "Executing sysinit.$NODE for node $NODE (server) ..."
echo
set -i
export TZ=est05edt04,M4.1.0/2,M10.5.0/2 # Eastern standard time zone
rtc -l hw                               # Set time from hardware clock
Dev &                                   # Start terminal device manager
Dev.ansi -Q -n 6 &                      # Start console driver in QNX emulation
reopen //0/dev/con1                     # Attach standard I/O to console
Dev.ser &                               # Start serial device driver
Dev.par &                               # Start parallel device driver

#
# If you have problems with mouse input, start the specific mouse driver for
# your mouse.  We have seen problems where mouse click input stops after the
# driver sees a double click and starts again once the mouse is moved.  This
# problem goes away when the specific mouse driver is started.
#

mousetrap start &                       # Autodetect mouse and start driver
Fsys.floppy &                           # Floppy disk controller driver
Pipe &                                  # Start the pipe manager
emu87 &                                 # 80x87 floating piont emulator

#
# This sysinit assumes the network card is at I/O address 0x320 and IRQ 5.
# If your network card is at a different I/O address or IRQ, change the
# -p and -i options on Net.ether1000.  Don't forget to disable plug and
# play and reserve the network card IRQ in the BIOS setup (if the BIOS
# setup supports this option).
#
# The I/O address and IRQ for the machine trying to boot from the network
# are specified in the /boot/build/ws.ether1000 file (or the startup file
# specified in /etc/config/netboot; This startup file is used by nodes
# booting off the network to build their OS on the fly).
#
# The following is a summary of the steps necessary to set up a QNX network
# consisting of a single server and multiple clients that boot off the server.
```

```
# The server has a hard disk, the clients do not have a hard disk.
#
# 1) Install QNX licenses for each machine on the server.
# 2) Edit /etc/config/netmap and add entries for each machine.
#    You must know the physical ID of each network card.  Make sure the
#    network cards are jumpered properly and the I/O addresses and IRQ's
#    are the same as those specified on the Net.ether1000 command line.
# 3) Create a sysinit.n for each machine on the network, where n is the
#    Node ID of the particular machine.  The server should be assigned
#    node 1 and all other machines should be assigned consecutive Node
#    ID's starting at 2.  The sysinit.server file may be used as an
#    example for sysinit.1 and the sysinit.client_wo_disk may be used
#    as an example for sysinit.n, where n is the node ID of a machine.
# 4) Connect the server and clients with the appropriate ethernet cables.
# 5) Reboot the server.
# 6) Reboot the client.
#

Net &                                   # Start the network manager
# Net.ether1000 -l 1 -p 320 -i 5 &        # NE1000/2000 ethernet network driver
Net.ether1000 -l 1 -p 240 -i 9 &        # NE1000/2000 ethernet network driver
sleep 1                                 # Give it a second to start
netmap -f                               # Load logical=>physical node ID table
netboot -B 18 &                         # Service boot requests from the network

#
# Network activity increases exponentially as more nameloc's are added to
# a networked system.  Therefore, a machine which is unlikely to go down
# should be selected to run nameloc for the entire network.  All other
# machines should not start up nameloc.  The server should run nameloc,
# the client should not.
#

nameloc &                               # Name locator

#
# The DOS filesystem manager is only needed if you have a DOS/Windows
# partition on the disk and would like to transfer files between the
# QNX and DOS partitions (this is the case when IsaGraph is installed
# on the DOS/Windows partition).
#

Dosfsys > /dev/null &                   # Start the DOS filesystem manager

#
# Starting the windows drivers in the sysinit file reduces the length
# of time it takes to start OpenCNC the first time.
#

/usr/lib/windows/bin/gr.vga &           # VGA 16 color graphics driver
/usr/lib/windows/bin/qwslib             # 16-bit Windows shared object library
/usr/lib/windows/bin/qwslib32           # 32-bit Windows shared object library

tinit -T /dev/con* -t /dev/con1 &       # Initialize terminals
```

Appendix E

sysinit.2

```
#
# Example sysinit for a client machine with a disk
#

echo
echo "Executing sysinit.$NODE for node $NODE (client with disk) ..."
set -i
export TZ=est05edt04,M4.1.0/2,M10.5.0/2 # Eastern standard time zone
rtc -r 100 -S 60 net 1                  # Set time from Node 1 on the network
Dev &                                   # Start terminal device manager
Dev.ansi -Q -n 6 &                      # Start console driver in QNX emulation
reopen //0/dev/con1                     # Attach standard I/O to console
Dev.ser &                               # Start serial device driver
Dev.par &                               # Start parallel device driver

#
# If you have problems with mouse input, start the specific mouse driver for
# your mouse.  We have seen problems where mouse click input stops after the
# driver sees a double click and starts again once the mouse is moved.  This
# problem goes away when the specific mouse driver is started.
#

mousetrap start &                       # Autodetect mouse and start driver
Fsys.floppy &                           # Floppy disk controller driver
Pipe &                                  # Start the pipe manager
emu87 &                                 # 80x87 floating piont emulator

#
# This sysinit assumes the network card is at I/O address 0x320 and IRQ 5.
# If your network card is at a different I/O address or IRQ, change the
# -p and -i options on Net.ether1000.  Don't forget to disable plug and
# play and reserve the network card IRQ in the BIOS setup (if the BIOS
# setup supports this option).
#

Net &                                   # Start the network manager
Net.ether1000 -l 1 -p 320 -i 5 &        # NE1000/2000 ethernet network driver


#
# Network activity increases exponentially as more nameloc's are added to
# a networked system.  Therefore, a machine which is unlikely to go down
```

```
# should be selected to run nameloc for the entire network.  All other
# machines should not start up nameloc.
#

nameloc &                               # Name locator

#
# Start the Touch screen driver.  Starting in release 2.0b5, a new Touch
# has been added to the QNX Install.  This Touch processes input differently
# than the old Touch.  The old Touch and TCalib have been moved to
# /bin/Touch.old and /bin/TCalib.old, and the following sysinit lines
# should be deleted from the sysinit file:
#
# stty baud=2400 par=none bits=7 stopb=2 < /dev/ser1
# Touch -t 1 -s 2400 mtouch < /dev/ser1 &
#
# The lines for the new Touch follow (You must choose between a MicroTouch
# or ELO touchscreen):
#
# Touch &
# Input -d/dev/touchin microtch -b 2400 fd -d/dev/ser1 abs & # MicroTouch
# Input -d/dev/touchin smartset fd -d/dev/ser1 abs &         # ELO
#
# If you have problems with the new Touch and need to go back to the old one
# (MicroTouch only - old Touch did not work with an ELO touchscreen), comment
# out the New Touch lines, uncomment the old Touch lines, and execute the
# following commands:
#
#   cd /bin
#   cp Touch Touch.new
#   cp TCalib TCalib.new
#   cp Touch.old Touch
#   cp TCalib.old TCalib
#
# /usr/local/bin/NewTouch contains Tech Notes on the new Touch, including
# instructions for calibrating both the MicroTouch and ELO touchscreens.

Touch &
Input -d/dev/touchin microtch -b 2400 fd -d/dev/ser1 abs & # MicroTouch

#
# Starting the windows drivers in the sysinit file reduces the length
# of time it takes to start OpenCNC the first time.
#

/usr/lib/windows/bin/gr.vga &           # VGA 16 color graphics driver
/usr/lib/windows/bin/qwslib             # 16-bit Windows shared object library
/usr/lib/windows/bin/qwslib32           # 32-bit Windows shared object library

tinit -T /dev/con* -t /dev/con1 &       # Initialize terminals
```

Appendix F

.profile

```
export ENV=$HOME/.kshrc
#export noclobber
export PAGER=less

PATH=:.:$HOME/bin$PATH
# PATH=:.:/usr/mdsi/2.0/bin:/usr/mdsi/2.0/kl:/home/kl$PATH
PATH=:.:/usr/mdsi/2.0/bin:/usr/mdsi/2.0/machines/bridge:/home/bridge$PATH

export PS1='{$LOGNAME}: '

umask 022
# aliases must appear in the .kshrc file

echo " OpenCNC 2.01 Software "
echo " Type 'runbridge' to start OpenCNC on machine tool bgp1:  "
```

Appendix G

startbgp1

```
m 1 # Machine ID number
a /usr/mdsi/2.0/bin/accMsg # get accMSg program
s /usr/mdsi/2.0/bin/prioSyms
# s /usr/mdsi/2.0/bin/prioSyms # read program priority symbols file
s /usr/mdsi/2.0/machines/bridge/dioSymsbgp1
# s /usr/mdsi/2.0/machines/bridge/dioSymsbgp1 # load DIO symbols file
p /usr/mdsi/2.0/bin/isaCalc -dio /usr/mdsi/2.0/machines/bridge/bgp1dio -msg /usr/mdsi/2.0/machines/bridge/bgp1msg
# start isaCalc and read softPLC and message files
p /usr/mdsi/2.0/bin/intCalcTVGO # start program hardware drivers
p /usr/mdsi/2.0/bin/movCalc -j 0 # start interpolater for job stream 1
p /usr/mdsi/2.0/bin/rpmCalc # start RPM PID Loop
p /usr/mdsi/2.0/bin/seqCalc -j 0 # start sequencer for job stream 1
b # get shared memory
t /usr/mdsi/2.0/machines/bridge/tunebgp1
#t /usr/mdsi/2.0/machines/bridge/tunebgp1 # read accMsg part of Tune file
g # release all accMsg (programs (go!)
a /usr/mdsi/2.0/bin/conMsg # get conMsg program
s /usr/mdsi/2.0/bin/prioSyms
# s /usr/mdsi/2.0/bin/prioSyms # read priority symbols file
p /usr/mdsi/2.0/bin/blkCalc # start parser for job stream
p /usr/mdsi/2.0/bin/oprDisp # start operator dispatch
p /usr/mdsi/2.0/bin/oprOff # start offset manager
p /usr/mdsi/2.0/bin/oprProg # start program file manager
b # get shared memory
t /usr/mdsi/2.0/bin/stdGMMill
t /usr/mdsi/2.0/machines/bridge/tunebgp1
# t /usr/mdsi/2.0/bin/stdGMMill # read standard G & M codes file
# t /usr/mdsi/2.0/machines/bridge/tunebgp1 # read conMsg part of Tune file
g # release conMsg programs (go!)
q /usr/mdsi/2.0/bin/winPad # displays touch-screen alphanumeric
# or numeric keypad
```

Appendix H

dioSymsbgp1

```
# dioSymsbgp1
# dioSymsbgp1 defines the digital inputs and outputs on the Western Reserve
# Control card with the variable names used bgp1dio and bgp1msg

# Digital Inputs

0 DI_X_REF
1 DI_X_NOTOT
2 DI_Y_REF
# 3 DI_EMPTY1
4 DI_Z_REF
5 DI_Z_NOTOT
6 DI_LUBE_NOTLOW
7 DI_AIR_NOTLOW
8 DI_SPINDLE_RDY
9 DI_LOWGEAR
10 DI_WATCHDOG_FAIL
11 DI_X_READY
12 DI_Y_READY
13 DI_Z_READY
14 DI_SERVO_ENERGIZ
15 DI_Y_NOTOT
16 DI_CYCLE_START
17 DI_FEEDHOLD # Not wired
18 DI_RESET_PB # Not wired

# Digital Outputs
# DO_COOLNT_MIST and DO_COOLNT_SPRAY are not wired yet.

0 DO_SSINC
1 DO_SSDEC
2 DO_NOT_BRAKE
# 3 DO_EMPTY2
4 DO_SPINDLE_CW
# 5 DO_EMPTY1
6 DO_SERVO_PULSE
7 DO_SPINDLE_CCW
8 DO_SERVO_HOLD
9 DO_COOLNT_MIST
16 DO_LUBE_PMP
18 DO_COOLNT_SPRAY
```

Appendix I

tunebgp1

```
#tunebgp1 --- Bridgeport


[accMsg]

# defFakeInterrupt   1 # Comment out for a real machine
# defFakeMachine     1 # Comment out for a real machine


defAxisCount    3 # XYZ;012
jsAxisCount[0]     3 # XYZ;012
jsAxisBase[0]         0 # XYZ

defAxisLetters   XYZ
defIJKLetters   IJK

encRemap[0]  0  # X
encRemap[1]  2  # Y
encRemap[2]  1  # Z
encRemap[3]  3  # Spindle Encoder (Formerly unused)
encRemap[4]  4  # unused
encRemap[5]  5  # unused

dacRemap[0]     0  # X
dacRemap[1]     1  # Y
dacRemap[2]     2  # Z
dacRemap[3]     3  # unused
dacRemap[4]     4  # unused
dacRemap[5]     5  # unused

#dacRemap[0]  0  # X
#dacRemap[1]  1  # Y
#dacRemap[2]  2  # Z
#dacRemap[3]  3  # unused (Was originally used to control the spindle)
#dacRemap[4]  4  # unused
#dacRemap[5]  5  # unused
# dacRemap[6] 6  # unused
# dacRemap[7] 7  # unused

#defDecimalShift  0
defInchModeDeflt  1 # machine comes up in inch mode
defRPMLkPstTlchg  1
```

```
dioCountInputs  3 # no. of input  groups (8 pts. per group)
dioCountOutputs  3 # no. of output groups (8 pst. per group)
dioAddressInput[0]   0x8108
dioAddressInput[1]   0x8109
dioAddressInput[2]  0x810A
dioAddressOutput[0]   0x810B
dioAddressOutput[1]  0x810C
dioAddressOutput[2]  0x810D
dioAddressInit[0]   0x810F

# dioZMinIndex   20320000 # 80000

defIntPerSec    512 # 512 interupts per second for servo loop
defDioCycleRate       64 # 64/512=1/8sec dio cycle time
defRPMCycleRate      64 # 64/512=1/8sec sec rpm rate


axFErrGain[0]      17000 # Originally 40000. 1000 for testing purposes
axFErrGain[1]      17000 # Originally 40000. 1000 for testing purposes
axFErrGain[2]      15000 # Originally 30000. 1000 for testing purposes

dacVoltMax[0]  2047 #0x7FF
dacVoltMax[1]  2047  #0x7FF
dacVoltMax[2]  2047 #0x7FF

defJerkMax    20 #  12
defAccelMax          150 # 300

axVelMax[0]       4134 #  50 IPM
axVelMax[1]       4134 #  50 IPM
axVelMax[2]       4134 #  50 IPM
axVelMaxSlow[0]    827 #  10 IPM
axVelMaxSlow[1]    827 #  10 IPM
axVelMaxSlow[2]    827 #  10 IPM


jsVelocityDryRun[0]  16536 # 200 IPM

axFErrMaxAllowed[0]   8192
axFErrMaxAllowed[1]   8192
axFErrMaxAllowed[2]   8192

axRefAtNegEnd[0] 1
axRefAtNegEnd[1] 1
axRefAtNegEnd[2] 0

axLocHome[0]     2540000 # 1.000"
axLocHome[1]        2540000 # 1.000"
# axLocHome[2]              0 # 0.000"
axLocHome[2]         11430000 # 4.500"
axLocAbsMin[0]    1905000 # 0.800"
axLocAbsMin[1]    1905000 # 0.800"
# axLocAbsMin[2]  -10160000 # -4.000"
axLocAbsMin[2]      0 # 0.000"
axLocAbsMinSlow[0]    2540000 #  0.800"
axLocAbsMinSlow[1]    2540000 #  0.800"
# axLocAbsMinSlow[2]  -10160000 # -4.000"
axLocAbsMinSlow[2]    0 # 0.000"
axLocAbsMax[0]       63500000 # 25.000"
axLocAbsMax[1]      304800000 # 12.000"
```

```
# axLocAbsMax[2]     127000 #  0.050"
axLocAbsMax[2]     11430000 #  4.500"
axLocAbsMaxSlow[0]   63500000 # 25.000"
axLocAbsMaxSlow[1]   30480000 # 12.000"
# axLocAbsMaxSlow[2]    1270000 #  0.000"
axLocAbsMaxSlow[2]    11430000 #  4.500"

axSyncOffset[0]    0 #
axSyncOffset[1]    0 #
axSyncOffset[2]    0 #
#axSyncOffset[0]    2540000 #  1.000"
#axSyncOffset[1]    2540000 #  1.000"
#axSyncOffset[2]    2260600 #  0.890"

axBackLash[0]      2777 #  0.0011"
axBackLash[1]     11000 #  0.0040"
axBackLash[2]     20320 #  0.0080"

encCountPerRev[0]   1000
encCountPerRev[1]   1000
encCountPerRev[2]   1000
encCountPerRev[3]   4096

# Pitch of the drive screws
axConvertDist[0]     508000 # 0.2000in/rev*2540000
axConvertDist[1]     508000 # 0.2000in/rev*2540000
axConvertDist[2]     508000 # 0.2000in/rev*2540000

# Encoder ticks per screw revolution (pulses= 4 x encoder lines)
axConvertTicks[0]   2000 # 1000pulses/encrev*2encrev/screwrev
axConvertTicks[1]   2000 # 1000pulses/encrev*2encrev/screwrev
axConvertTicks[2]   2000 # 1000pulses/encrev*2encrev/screwrev

defCheckBits     0x0

spDeltaPercent     10
spRPMNearZeroVal     60 # Originally was 10

# Ask Todd or Chuck about dio variables.  They are not in the Axis Variable manual.
encSpindleNum       3 # Encoder channel for the spindle
# dacSpindleNum       3
#spRPMNearZeroVal   50
spConvertRPM      480 # 60sec/min*8interupts/sec=480interupts/min
spConvertTicks  28672 # Encoder pulses per revolution
spDeltaPercent     10
spSimulateK         0 # Were are running closed loop

# dacSpindleNum   3
# defRPMOpenLoop 1 # If no encoder set this to 1 to indicate spindle is running open loop.
# spSimulateK        0x40000000 # If no encoder this is a simulated feedback gain


dioRpmSwitch     40 #
# dioRPMNearZ[0] 50
# dioConvertT[0] 28672
dioMaxRpm[0]    480 # Low gear max rpm
dioMinRpm[0]     60 # Low gear min rpm
dioKRpm[0] 48000
dioKSum[0]  0400
dioSumLim[0]     100
# dioRPMNearZ[1] 500
```

```
# dioConvertT[1] 4096
dioMaxRpm[1]   4280 # High Gear max rpm
dioMinRpm[1]    490 # High Gear min rpm
dioKRpm[1] 48000
dioKSum[1]  0400
dioSumLim[1]     100

dioEncMaxSync        350

dioLubeOnTime         5 # seconds
dioLubeOffTime        45 # seconds

[conMsg]

jsToolRadOffDisp[0]   1

# not in Axis Variable Dictionary
axToolLenOffDisp[0]  1
axToolLenOffDisp[1]  1
axToolLenOffDisp[2] 1

axFixtureDisplay[0]     1
axFixtureDisplay[1]     1
axFixtureDisplay[2]     1

defFixTLOffImmed        1 # 0=deferred action; 1=immediate
defFix1WithOff  0 # fixture 1 on with any offset

defLocOptionsOff     1
defParseComments  1

# not in Axis Variable Dictionary
defGCirCCW  2
defGCirCW  3

defProgFilePath   /home/bridge/program/
defProgFileName   /home/bridge/program/pray1.ncc
defTLOffFilePath  /home/bridge/offset/
defTLOffFileName  /home/bridge/offset/ToolOff0
defFixFilePath      /home/bridge/fixture/
defFixFileName /home/bridge/fixture/FixOff0
```

Appendix J

bgp1.gen

```
# bgp1.gen

# OpenCNC(TM)
# Manufacturing Data Systems, Inc.(TM)
# http://www.mdsi2.com
# Copyright 1993-1997
#
# $Id$
#
// This is the generator file for a Bridgeport machine.
//
// Messages handled by ISaGRAF message application:
// (DIOM_ and DIOS_ symbols defined by the system, usable here)
// (DIOU_ symbols used in ISaGRAF application, not usable here)
//
// Message type, parameters     Function
// ------------------------ --------
// DIOM_MISCFUNC(1),
//   ss_3 dioSpnDirReq=2
//   ss_4 dioSpnDirReq=1
//   ss_7 coolant on mist
//   ss_8 coolant on flood
//   ss_9 coolant off
//   ss_12 tool index reverse
//   ss_13 tool index forward
//   ss_14 open chuck bar feed
//   ss_15 close chuck bar feed
//   ss_41 low range
//   ss_42 high range
//   ss_50 conveyer on
//   ss_51 conveyer off
// DIOM_SYSTEM(2),
//   DIOS_CSCHECK(1) returns ss_0 (fail), or ss_1 (success)
//   DIOS_EOP(2) end of program
//   DIOS_JOG(3),sh,sh,lo type,axis,DIST
//   DIOS_JOG2RUN(4)
//   DIOS_OSTOP(5) ignored, null reply
//   DIOS_RUN2JOG(6)
//   DIOS_SEQCALCRST(7),sh reset sequencer for JID 'sh'
//   DIOS_SEQCALCRUN(8),sh clear stop bit in sequencer for JID 'sh'
//   DIOS_SOFTSTART(9) soft start
//   DIOS_STOP(10) auto stop, M00, M01
//   DIOS_STOPS(11) ignored, null reply
//   DIOS_TLCHG(12),lo tool change to value 'lo'
//   DIOS_TLSEL(13),lo ignored, null reply
// DIOM_USER(3),
//   ss_1(DIOU_SYNC)      sync machine ("Sync")
//   ss_2(DIOU_ENERGIZE),sh 0="AxesStop", 1="AxesStart"
//   ss_3(DIOU_CHUCK),sh 1=open 2=close
//   ss_5(DIOU_FEEDHOLD) "JogAbsX","JogAbsY","JogAbsZ","JogHome"
//   ss_6(DIOU_CYCLESTART) cycle start
//   ss_7(DIOU_JOGMODE) jog mode "Jog"
//   ss_8(DIOU_RUNMODE) run mode "Run"
//   ss_9(DIOU_SPNDLCCW) set spindle direction CCW
//   ss_10(DIOU_SPNDLCW) set spindle direction CW
//   ss_11(DIOU_SPNDLRUN) run spindle
//   ss_13(DIOU_TOOLDECR) decrement tool num
//   ss_14(DIOU_TOOLINCR) increment tool num
```

```
//   ss_15(DIOU_TOOLNUMSET),lo set tool num to value 'lo'
//   ss_16(DIOU_TUROFFSETRD) "SetTurOffset" -> dioTurOffset
//   ss_17(DIOU_TUROFFSETWT) "SetTurOffset" <- dioTurOffset
//   ss_18(DIOU_RPMSET),sh set new RPM value
//
// $Log$
////////////////////////////////////////////////////////////////////////////////



[winGen]

TITLE("MDSI Operator Panel - BGP1");

// ------------------------------------ S P I N D L E
RECT( 1 , 1 ){

  NOSELECTONFLAG(ss_mRunMode);


  LINE("Req RPM $$$$       Range    $$$$", ss_mRPMValue, ss_spRPMMax);
  LINE("Cur RPM $$$$ <           > $$$$", ss_mRPMNow, <ss_mRPMNow, ss_spRPMMax >, spnRunText);
  LINE("Act RPM $$$$ <           > $$$$", ss_spRPMState, <ss_spRPMState, ss_spRPMMax>, spnDirText);


  MENU( "StopSpndl",   ASSIGN(sl_dioSpnDirRequest, sl_dioSpnAutoStop);
     "StopFree",    ASSIGN(sl_dioSpnAutoStop, 0): ASSIGN(sl_dioSpnDirRequest, 0);
     "StopLock",    ASSIGN(sl_dioSpnAutoStop, 3): ASSIGN(sl_dioSpnDirRequest, 3);

    // The character '*' that follows is used to keep menu displayed, don't close

     *"SetRpm",      MSG(DIOM_USER, ss_18, PROMPTSHORT("Enter new RPM", ss_mRPMValue, 0, 9999) );
     *"SetSpndlCw",  MSG(DIOM_USER, ss_10);
     *"SetSpndlCcw", MSG(DIOM_USER, ss_9);
      "RunSpndl",     MSG(DIOM_USER, ss_11);
      );

  TEXT( sl_dioSpnAutoDir,  0, "FREE", spnDirText;
sl_dioSpnAutoDir,  1, "CW",   spnDirText;
sl_dioSpnAutoDir,  2, "CCW",  spnDirText;
sl_dioSpnAutoDir,  3, "LOCK", spnDirText;
sl_dioSpnDirState, 0, "FREE", spnRunText;
sl_dioSpnDirState, 1, "CW",   spnRunText;
sl_dioSpnDirState, 2, "CCW",  spnRunText;
sl_dioSpnDirState, 3, "LOCK", spnRunText;
  );


  COLOR(DEFAULT, RED;
c_genRpmGreen, 1, GREEN;
   );}


//------------------------------ C O O L A N T   F L O O D

RECT( 1 , 6 ){


  LINE("^Coolant  $$$$ ", coolflManText);
  LINE("^ Flood    $$$ ", coolflText);


  MENU( "Off",       ASSIGN(c_dioCoolflManual, 1):ASSIGN(sl_dioCoolflReq, 0);
"On",        ASSIGN(c_dioCoolflManual, 1):ASSIGN(sl_dioCoolflReq, 1);
"AutoOff",   ASSIGN(c_dioCoolflManual, 0):ASSIGN(sl_dioCoolflReq, 0);
"AutoOn",    ASSIGN(c_dioCoolflManual, 0):ASSIGN(sl_dioCoolflReq, 1);
      );


  TEXT( c_dioCoolflManual, 0, "AUTO", coolflManText;
c_dioCoolflManual, 1, "    ", coolflManText;
sl_dioCoolflState, 0, "OFF",  coolflText;
sl_dioCoolflState, 1, "ON ",  coolflText;
  );


  COLOR(DEFAULT, RED;
c_dioCoolflManual, 0, GREEN;
   );}
```

```
//------------------------------ C O O L A N T  M I S T

RECT( +18 , +0 ){


  LINE("^Coolant  $$$$ ", coolmiManText);
  LINE("^ Mist     $$$ ", coolmiText);


  MENU( "Off",       ASSIGN(c_dioCoolmiManual, 1):ASSIGN(sl_dioCoolmiReq, 0);
"On",        ASSIGN(c_dioCoolmiManual, 1):ASSIGN(sl_dioCoolmiReq, 1);
"AutoOff",   ASSIGN(c_dioCoolmiManual, 0):ASSIGN(sl_dioCoolmiReq, 0);
"AutoOn",    ASSIGN(c_dioCoolmiManual, 0):ASSIGN(sl_dioCoolmiReq, 1);
      );


  TEXT( c_dioCoolmiManual, 0, "AUTO", coolmiManText;
c_dioCoolmiManual, 1, "    ", coolmiManText;
sl_dioCoolmiState, 0, "OFF",  coolmiText;
sl_dioCoolmiState, 1, "ON ",  coolmiText;
  );


  COLOR(DEFAULT, RED;
c_dioCoolmiManual, 0, GREEN;
   );}


//------------------------------    O V E R  R I D E S
RECT( 1 , 9 ){

LINE("Rpm%");
   SLIDE(ss_spPctOvRide, 1, 0, 200, "<<< SLOW OVERRIDE RPM >>>");}

RECT( +5 , +0 ){

   LINE( "$$$ <                  > ", ss_spPctOvRide, <ss_spPctOvRide, 200 >);
   SLIDE(ss_spPctOvRide, 0, 0, 200, "<<< OVERRIDE RPM >>>");}

RECT( 1 , +2 ){

   LINE("Fpm%");
   SLIDE(ss_mPctFeedOvRide, 1, 0, 200, "<<< SLOW OVERRIDE FPM >>>");}

RECT( +5 , +0 ){

   LINE( "$$$ <                  > ", ss_mPctFeedOvRide, <ss_mPctFeedOvRide, 200 >);
   SLIDE(ss_mPctFeedOvRide, 0, 0, 200, "<<< OVERRIDE FPM >>>");}

RECT( 1 , +2 ){

   LINE("Rap%");
   SLIDE(ss_mPctRapidOvRide, 1, 0, 100, "<<< SLOW OVERRIDE RAPID >>>");}

RECT( +5 , +0 ){

   LINE( "$$$ <          >", ss_mPctRapidOvRide, <ss_mPctRapidOvRide, 100 >);
   SLIDE(ss_mPctRapidOvRide, 0, 0, 100, "<<< OVERRIDE RAPID >>>");}

RECT( 1 , +2 ){

   LINE("Jog%");
   SLIDE(ss_mPctJogOvRide, 1, 0, 100, "<<< SLOW OVERRIDE JOG >>>");}

RECT( +5 , +0 ){

   LINE( "$$$ <          >", ss_mPctJogOvRide, <ss_mPctJogOvRide, 100 >);
   SLIDE(ss_mPctJogOvRide, 0, 0, 100, "<<< OVERRIDE JOG >>>");}


//--------------------------------    O P T I O N A L  S T O P
RECT( 1 , 17 ){

  LINE(" OpStop   $$$$ ", opStopText);


  MENU( "Off",      ASSIGN(ss_mOptionalStopOn, 0);
    "On",       ASSIGN(ss_mOptionalStopOn, 1);
    "LockedOn", ASSIGN(ss_mOptionalStopOn, 2);
```

```
      );

  TEXT( ss_mOptionalStopOn, 2, "LOCK",  opStopText;
ss_mOptionalStopOn, 1, " ON ",  opStopText;
ss_mOptionalStopOn, 0, " OFF",  opStopText;
      );

  COLOR(DEFAULT, RED;
ss_mOptionalStopOn, 0, GREEN;
   );}


//------------------------------------ S I N G L E   B L O C K
RECT( +18 , +0 ){


  LINE( "Single Blk $$$",singleBlkText);


  MENU( "Off", ASSIGN(ss_mSingleBlockOn, 0);
"On",  ASSIGN(ss_mSingleBlockOn, 1);
      );


  TEXT( ss_mSingleBlockOn, 1, "ON ",  singleBlkText;
ss_mSingleBlockOn, 0, "OFF", singleBlkText;
  );


  COLOR(DEFAULT, RED;
     ss_mSingleBlockOn, 0, GREEN;
  );}


//---------------------------------------- M O D E
RECT(  1 , 20 ){


  LINE( "Mode  $$$",   cmdModeText);
  LINE( "         ");


  MENU( "Jog", MSG(DIOM_USER, ss_7): ASSIGN(us_dioJogType, 0): ASSIGN(sl_dioJogDist, 0);
"Run", MSG(DIOM_USER, ss_8): ASSIGN(us_dioJogType, 0): ASSIGN(sl_dioJogDist, 0);
  );

  TEXT( ss_mRunMode,   1, "RUN", cmdModeText;
ss_mRunMode,   0, "JOG", cmdModeText;
  );}


//------------------------------------ A X E S   J O G
RECT( +11 , +0 ){

  HIDEONRUN;

  LINE(" Axes     ");
  LINE(" Jog #####", jogTypeText);


  MENU( # start all items with DIOS_SEQCALCRST

    "Off",      MSG(DIOM_SYSTEM, DIOS_SEQCALCRST, ss_0):
                CLOSEWIN:
                ASSIGN(us_dioJogType, 0):
                ASSIGN(sl_dioJogDist, 0);
    "Slide",    MSG(DIOM_SYSTEM, DIOS_SEQCALCRST, ss_0):
                ASSIGN(us_dioJogType, 1):
                ASSIGN(sl_dioJogDist, 0):
         SELECTSLIDE(ss_axJogRate, 0, -100, 100);
    "Delta",    MSG(DIOM_SYSTEM, DIOS_SEQCALCRST, ss_0):
MENU(   "SetDelta", SELECTDELTA(PROMPTDIST("Enter new DELTA:", sl_dioJogDist, 100, 10000000));
    "1.0000",   SELECTDELTA(1.0);
    "0.1000",   SELECTDELTA(0.1);
    "0.0100",   SELECTDELTA(0.01);
    "0.0010",   SELECTDELTA(0.001);
    "0.0001",   SELECTDELTA(0.0001);
    );
    "JogAbsX",  MSG(DIOM_SYSTEM, DIOS_SEQCALCRST, ss_0):
                ASSIGN(sl_genTempLong, PROMPTDIST("Enter new AbsX")):  CLOSEWIN:
ASSIGN(us_dioJogType, 0):  ASSIGN(sl_dioJogDist, 0):
         MSG(DIOM_USER, ss_5): # DIOU_FEEDHOLD
         MSG(DIOM_SYSTEM, DIOS_JOG, ss_0, ss_1, sl_genTempLong);
```

```
# type = ss_0 = absolute jog
# axis = ss_1 = 1 << 1 = X axis
    "JogAbsY", MSG(DIOM_SYSTEM, DIOS_SEQCALCRST, ss_0):
                ASSIGN(sl_genTempLong, PROMPTDIST("Enter new AbsY")):  CLOSEWIN:
ASSIGN(us_dioJogType, 0):  ASSIGN(sl_dioJogDist, 0):
        MSG(DIOM_USER, ss_5): # DIOU_FEEDHOLD
        MSG(DIOM_SYSTEM, DIOS_JOG, ss_0, ss_2, sl_genTempLong);
# type = ss_0 = absolute jog
# axis = ss_2 = 1 << 2 = Y axis
    "JogAbsZ", MSG(DIOM_SYSTEM, DIOS_SEQCALCRST, ss_0):
                ASSIGN(sl_genTempLong, PROMPTDIST("Enter new AbsZ")):  CLOSEWIN:
ASSIGN(us_dioJogType, 0):  ASSIGN(sl_dioJogDist, 0):
        MSG(DIOM_USER, ss_5): # DIOU_FEEDHOLD
        MSG(DIOM_SYSTEM, DIOS_JOG, ss_0, ss_4, sl_genTempLong);
# type = ss_0 = absolute jog
# axis = ss_3 = 1 << 4 = Z axis
    "JogHome",  MSG(DIOM_SYSTEM, DIOS_SEQCALCRST, ss_0):  CLOSEWIN:
ASSIGN(us_dioJogType, 0):  ASSIGN(sl_dioJogDist, 0):
MSG(DIOM_USER, ss_5): # DIOU_FEEDHOLD
MSG(DIOM_SYSTEM, DIOS_JOG, ss_0, ss_4, sl_axLocHome[2]):
# type = ss_0= absolute jog
# axis = ss_3=1  Z 1st
MSG(DIOM_SYSTEM, DIOS_JOG, ss_0, ss_2, sl_axLocHome[1]):
# type = ss_0= absolute jog
# axis = ss_2=1  Y 2nd
MSG(DIOM_SYSTEM, DIOS_JOG, ss_0, ss_1, sl_axLocHome[0]);
# type = ss_0= absolute jog
# axis = ss_1=1  X 3rd
       );

  TEXT( us_dioJogType, 0,      "  OFF", jogTypeText;
us_dioJogType, 1,       "SLIDE", jogTypeText;
us_dioJogType, 2, sl_dioJogDist, jogTypeText;
  );

  COLOR(
    DEFAULT, RED;
    us_dioJogType, 0, GREEN;
    );
}


//--------------------------------   S T A R T U P
RECT( +13 , +0 ){


  HIDEONRUN;


  LINE("        ");
  LINE("        ");
  LINE(""StartUp ");
  LINE("        ");
  LINE("        ");


  MENU( "Sync",   CLOSEWIN:
  ASSIGN(us_dioJogType, 0): ASSIGN(sl_dioJogDist, 0): MSG(DIOM_USER, ss_1);
     "Machine Stop", CONFIRM("Stop Machine & Axes -- OK??"):
ASSIGN(us_dioJogType, 0): ASSIGN(sl_dioJogDist, 0): MSG(DIOM_USER, ss_2, ss_0);
     "Machine Start",ASSIGN(us_dioJogType, 0): ASSIGN(sl_dioJogDist, 0): MSG(DIOM_USER, ss_2, ss_1);
  );

  COLOR(DEFAULT, BLINK;
    ss_mSyncDone, 1, GREEN;
   );}


//---------------------------------   C Y C L E   S T A R T
RECT(  1 , 24 ){


  LINE("         ");
  LINE("Cycle $$$", onoff);
  LINE("         ");

  MENU( "Start", MSG(DIOM_USER, ss_6); // DIOU_CYCLESTART
  );


  TEXT( c_genSeqORunning, 1, "ON",  onoff;
c_genSeqORunning, 0, "OFF", onoff;
```

```
  );


  COLOR(DEFAULT, RED;
c_genSeqORunning, 1, GREEN;
   );}

//--------------------------------------    F E E D H O L D
RECT( +11 , +0 ){


  LINE(" Feed     ");
  LINE(" Hold  $$$", feedText);
  LINE("          ");

  MENU( "Hold", MSG(DIOM_USER, ss_5);
  );


  TEXT( ss_mFeedHold, 1, "ON ", feedText;
ss_mFeedHold, 0, "OFF", feedText;
  );


  COLOR(DEFAULT, GREEN;
ss_mFeedHold, 1, RED;
   );}

//--------------------------------------    E N D
//--------------------------------------    E N D
//--------------------------------------    E N D
```

Appendix K

bgp1.men

```
0:Startup:GROUPSTART
0:Start OpenCNC:startup:-f /usr/mdsi/2.0/machines/bridge/startbgp1
0:Operator Panel:winGen:@-f /usr/mdsi/2.0/machines/bridge/bgp1.gen
0:Message:winMsg:@
0:Position Display:winPos:@
0:Program Display:winProg:@-e
0:Startup:GROUPEND
0:Tool Offset Display:winOff:@
0:Fixture Offset Display:winFix:@
0:Calculator:/windows/apps/Calc/qwcalc:
1:Setup Programs:GROUPSTART
1:Program Editor:winProg:@
1:File Copy:winCopy:
1:OpenCNC Backup:/windows/bin/wterm:/home/bridge/bin/sysbackup.init
1:Setup Programs:GROUPEND
2:Tune Programs:GROUPSTART
2:WinVar:winVar:@
2:WinDioIn:winDioIn:@
2:WinDioOut:winDioOut:@-f
2:winFeShow:winFeShow:@
2:Tune Programs:GROUPEND
3:System Tools:GROUPSTART
3:System Terminal:/windows/bin/wterm:
3:Menu Maint Tool:/windows/bin/wterm:/home/bridge/bin/editmenu
3:TuneFile Editor:/windows/bin/wterm:/home/bridge/bin/edittune
3:WinDioOut w/Force:winDioOut:@-f
3:System Tools:GROUPEND
0:Stop Machine:MACHSTOP:
1:About...:ABOUT:
0:Authorization:AUTH:
0:Exit:EXIT:
1:7C6E7D5E7@:PASSWORD
1:7@6A7B7D:PASSWORD
2:4M6A6I6N7D:PASSWORD
2:7D7E6N6E:PASSWORD
3:6K6E6L6L7I3A:PASSWORD
3:7C7I7C7D6E6M:PASSWORD
3:7G6I6L6L:PASSWORD
3:6B6E6C6K7I:PASSWORD:James Franks
```

Vita

James Richard Franks was born July 16, 1974 in Chattanooga, Tennessee to Richard and Nancy Franks. After graduating from Ooltewah High School in 1992, he entered Tennessee Technological University and received his Bachelor of Science in Mechanical Engineering in 1996. The following year he began pursuing his Master of Science at the University of Tennessee at Knoxville and working in the Robotics and Electromechanical Systems Laboratory under the direction of Dr. William R. Hamel. After one year of graduate school, James married the lovely Rebecca Leigh Hill who was beginning her final year of law school. Upon graduation, James will begin his first engineering job working for Lexmark International Inc. in the Business Printer Development Division.