

## INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

**The quality of this reproduction is dependent upon the quality of the copy submitted.** Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

# U·M·I

University Microfilms International  
A Bell & Howell Information Company  
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA  
313/761-4700 800/521-0600



**Order Number 9209090**

**Implementation of a sensor-based supervision system for CNC  
machining**

**Wells, Robert Lindsay, Ph.D.**

**University of Florida, 1991**

**U·M·I**  
300 N. Zeeb Rd.  
Ann Arbor, MI 48106



IMPLEMENTATION OF  
A SENSOR-BASED SUPERVISION SYSTEM  
FOR CNC MACHINING

By

ROBERT LINDSAY WELLS

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL  
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT  
OF THE REQUIREMENTS FOR THE DEGREE OF  
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

1991

## ACKNOWLEDGEMENTS

The author would like to express his sincere gratitude to Dr. Jiri Tlusty and Dr. Scott Smith for their guidance and generous support during this research. The author also thanks Dr. Carl Crane, Dr. John Schueller and Dr. Sencer Yeralan for serving on his committee, and Dr. Jose Principe for contributing his expertise to the project.

Special thanks go to Bob Winfough of the Machine Tool Laboratory, Russ Walters of the Electrical Engineering Department, John Frost of Manufacturing Laboratories and Gordie Hawes of Automation Intelligence for their assistance in the research.

Finally, to his parents, Bob and Connie Wells, the author sends his deepest love and gratitude for their unqualified support and encouragement during the long haul.

This research was sponsored in part by a National Science Foundation grant, #DDM-8914084, "Comprehensive Supervision System for Machining Centers."

TABLE OF CONTENTS

	<u>page</u>
ACKNOWLEDGEMENTS .....	ii
ABSTRACT .....	v
CHAPTERS	
1 INTRODUCTION .....	1
Sensor-Based Supervision of Machining .....	1
Review of the Literature .....	4
Outline of the Present Research .....	17
2 THE MACHINE TOOL AND ITS CONTROLLER .....	20
The White-Sunstrand Series 20 Omnimil .....	20
The Automation Intelligence Flexmate Controller .....	24
The FlexMate Motion Co-Processor .....	29
3 THE MACHINE MONITORING AND CONTROL SCHEMES .....	37
Adaptive Control System .....	37
The Fast Stopping Routine .....	40
Tool Breakage Detection System .....	42
Chatter Recognition and Control System .....	46
Spindle Torque Overload Detection System .....	50
4 THE ON-LINE MACHINE SUPERVISION SYSTEM .....	54
An Inventory of the Sensors .....	54
The Interface Hardware .....	59
The Interface Software .....	62
Integration of the Supervision System .....	64
5 THE OFF-LINE MACHINE EVALUATION SYSTEM .....	71
Hardware Requirements .....	71
Data Acquisition .....	75
Experimental Modal Analysis .....	79
Chatter Detection and Analysis .....	86

6	EXPERIMENTAL VERIFICATION OF THE ON-LINE SYSTEM .....	91
	The Fast Stopping Routine .....	91
	The Supervision Subroutines .....	96
	Adaptive Control System .....	101
	Tool Breakage Detection System .....	106
	Chatter Recognition and Control System .....	109
	Spindle Torque Overload Detection System .....	112
7	CONCLUSIONS AND RECOMMENDATIONS .....	116
APPENDICES		
A	LISTING OF THE SUPERVISION SOFTWARE .....	119
	Supervision Computer Interface .....	119
	FlexMate Motion Co-Processor Interface .....	127
	Fast Stopping Program .....	135
B	LISTING OF THE SPINDLE TORQUE OVERLOAD PROGRAM .....	138
C	LISTING OF THE MACHINE EVALUATION PROGRAM .....	145
	REFERENCES .....	178
	BIOGRAPHICAL SKETCH .....	183



Abstract of Dissertation Presented to the Graduate School  
of the University of Florida in Partial Fulfillment of the  
Requirements for the Degree of Doctor of Philosophy

IMPLEMENTATION OF  
A SENSOR-BASED SUPERVISION SYSTEM  
FOR CNC MACHINING

By

Robert Lindsay Wells

August 1991

Chairman: Dr. Jiri Tlustý  
Major Department: Mechanical Engineering

A sensor-based supervision system for CNC machining was implemented on a White Sunstrand Series 20 Omnimil machining center. An interface was created between the Automation Intelligence FlexMate CNC controller and a supervisory computer. A set of supervision subroutines was created that allowed monitoring and control schemes running on the supervisory computer to take control of the machine tool based on sensor data sampled during the metal cutting process.

Four supervision schemes were implemented as part of the on-line machine supervision system. The Adaptive Control scheme monitored the vibration of the metal cutting process and varied the feedrate to achieve stable cuts. The Tool Breakage Detection scheme stopped the cutting process when a broken tool was detected. The Chatter Recognition and Control

scheme detected chatter, a self-regenerative vibration phenomenon, and automatically selected a new stable spindle speed. The Spindle Torque Overload Detection scheme stopped the feed if a variation in the spindle speed indicated that the cutting tool was about to stall in the workpiece.

Also created was an off-line machine evaluation system which was used to measure Transfer Functions on the machine, sample data from sensors, and analyze machine tool chatter. The off-line system was used to evaluate the performance of both the supervision subroutines and the monitoring and control schemes in the on-line supervision system.

Each of the individual monitoring and control schemes was demonstrated in a series of cutting tests on the Omnimil. It was shown that the schemes were able to identify various disturbances in the cutting process, and take corrective action using the supervision subroutines to issue commands to the machine tool controller.

The research demonstrates the feasibility of having an external supervisory computer work in cooperation with a machine tool controller. This suggests that future CNC controllers should directly support this kind of interface.

## CHAPTER 1 INTRODUCTION

The issues that motivate sensor-based supervision of machine tools are discussed in this chapter, and a review of research into the monitoring and control of machining operations is presented. Adaptive control, broken tool detection, chatter recognition and control, and spindle torque overload detection will be emphasized. The original contributions of the present research, centering on the creation of a machine tool supervision system, are outlined.

### Sensor-Based Supervision of Machining

The principal reason for developing sensor-based supervision systems for the monitoring and control of Computer-Numerical Control (CNC) machine tools is to enhance their productivity. The objective is not necessarily to replace the machine operator, but to provide a fast and accurate response mechanism to disturbances in the milling process that limit the Metal Removal Rate (MRR), and which degrade the quality of the finished product.

After careful modeling and analysis of the dynamics of the milling process, sensor-based monitoring and control

schemes can be developed that permit the automatic detection and regulation of such phenomena as machine tool chatter and broken milling cutters. Individual schemes that measure, and sometimes control, several critical aspects of the milling process have been developed by many researchers over the years. Very little work has been done, however, in integrating these separate schemes into a comprehensive machine supervision system. It is the objective of the present research to implement such a comprehensive system.

A clear distinction needs to be drawn between sensor-based computer supervision of machining and Computer Integrated Manufacturing (CIM). The integration of Computer-Aided Design (CAD) facilities with the automatic generation of tool paths provided by Computer-Aided Manufacturing (CAM), and with production scheduling and inventory control programs, is called CIM. The trend towards full factory automation employs these tools with the aim of developing a computerized manufacturing environment.

Sensor-based supervision of production machinery, on the other hand, aims at increasing productivity and product quality by direct monitoring and control at the machine level. The supervision system may or may not be part of a larger CIM system. The machine tool phenomena that are most often monitored are tool condition, cutting forces and vibration, since these affect both MRR and product quality. A detailed

comparison of various sensing strategies will be presented in the next section.

The motivation for on-line monitoring and control of CNC machine tools is that operations such as the end milling of aluminum aircraft panels and the machining of cast iron auto body stamping dies require the removal of a significant percentage of the original metal (Smith and Delio, 1989). A high MRR is desirable for these time consuming and expensive operations, but the increased axial and radial tool immersions that are used for the higher MRR can cause the self-excited vibration phenomenon called chatter, can damage the cutting tool, or can cause poor surface finish. Sensing schemes that detect these events, and take corrective action, could provide significant increases in productivity and product quality. Automatic selection of optimally stable speeds in High Speed High Power (HSHP) milling (Delio et al., 1990), adaptive control of the machining feed rate to avoid chatter (Tlustý and Tyler, 1988) and the sensing of tool breakage (Tlustý and Tarny, 1988) are examples of proven schemes that could be included in a comprehensive supervision system.

The main barrier to the integration of sensor-based supervision systems, as well as CAD/CAM systems, with machine tool controllers is described by Wright et al. (1990a, p.322). They state that "in today's factories, the integration of such CAD tools and on-machine sensors is frustrated by the closed architecture of typical machine tool controllers." The problem

is that in order for the supervision system to act automatically upon sensed data, it must be able to take control of the machine tool, or be able to issue commands to the CNC controller. Creating the interface between a supervision system running on a remote computer and the CNC controller on the machine tool is not a trivial task.

The present research has taken advantage of the relatively open architecture of a commercially available machine tool CNC controller. However, the supervision system could be applied to any machine tool if access to the control parameters of the machining process were available.

#### Review of the Literature

The book by Pressman and Williams (1977) gives a thorough description of CNC machine tool technology as it is in common use today. The paper by Tlusty and Andrews (1983) and the report by Birla (1980) are often cited as excellent reviews of the sensors in common usage for monitoring and control of machining operations. Bolinger and Duffie (1988) also describe sensors in common use for the computer control of machines and processes. It can be seen from the dates of these references that sensor technology has for some time been adequate for the task of machine supervision. The challenge is to select sensors that monitor relevant aspects of the dynamics of the milling process, and to process their signals with algorithms

that provide robust fault detection. Past and present research into various machine tool monitoring and control schemes is described below, as well as the work of researchers who are attempting to integrate sensors into supervision systems.

Adaptive control (A/C) of machine tools has been researched for some time. Ulsoy et al. (1982) gave a comprehensive review of early work. When applied to machine tools, the term adaptive control has come to mean the variation of such machining parameters as feed rate and spindle speed based on sensed cutting data such as the cutting force or vibration. The objective is usually to maximize the MRR, or to prolong tool life.

Weck et al. (1975), whose work is described in more detail below, developed an A/C system that varied the spindle speed to avoid chatter. Lauderbaugh and Ulsoy (1986), whose paper also contains a good review of A/C research, designed a Model Reference Adaptive Control (MRAC) system for controlling the force in milling. They proposed regulating the feed rate based on a comparison of the cutting force (obtained from a dynamometer) to a dynamic model of the cutting process.

Tlusty and Tyler (1988) describe more recent efforts in the field, and outline an effective A/C scheme. The objective of their work was to both avoid chatter during cutting by varying the feed rate to keep the cutting force below a limiting value, and to detect when the tool impacts the workpiece. The latter capability allows the tool to be moved

at higher feedrates when it is not cutting, and the feed to be adjusted automatically to the correct cutting value when the high force of impact is sensed. This scheme will be implemented as part of the present research.

Figure 1.1 shows a simplified block diagram of an adaptive control system that is controlling the cutting force,  $F_a$ , by varying the feed rate,  $F$ . It can be seen that the machine tool and its controller are part of the control loop, and in this regard the system is similar to a machine tool supervision system. However, where adaptive control schemes usually aim at improving one aspect of machining (say, avoiding chatter), a supervision system would monitor and control all the relevant parameters of the machining operation. Figure 1.2 shows a block diagram of a comprehensive supervisory system that issues both speed and feed commands to the CNC controller based on sensed data.

One aspect of machining that is commonly monitored is the condition of the cutting tool. Both tool wear and tool breakage in milling, turning and drilling can be considered in these schemes. For catastrophic tool failure, the objective is to detect breakage with high reliability and a minimum of false alarms. A good review of research in on-line tool condition monitoring can be found in the paper by Johnson et al. (1988). These investigators are typical in their advocacy of using cutting force signals (obtained from dynamometers) and vibration signals of the tool relative to



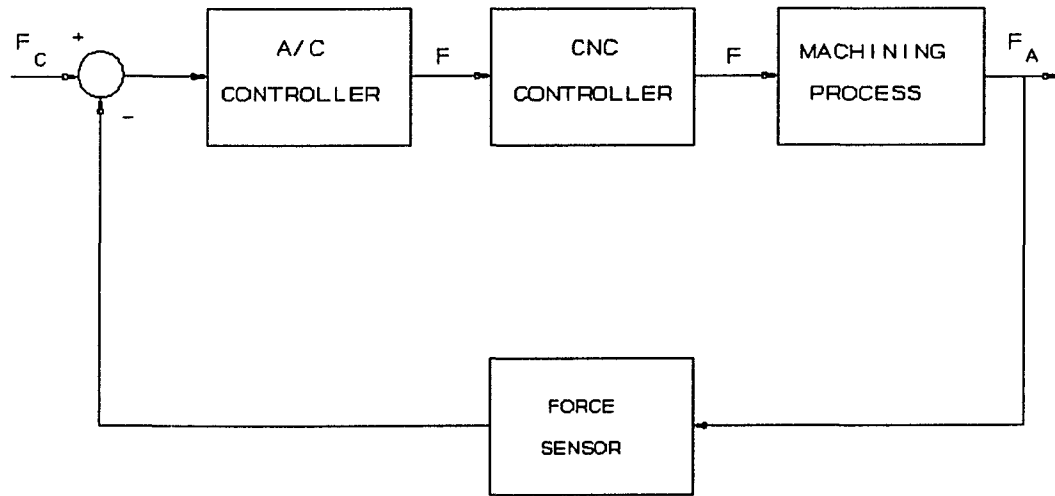


Figure 1.1. Block Diagram of an Adaptive Control System for a Machine Tool.

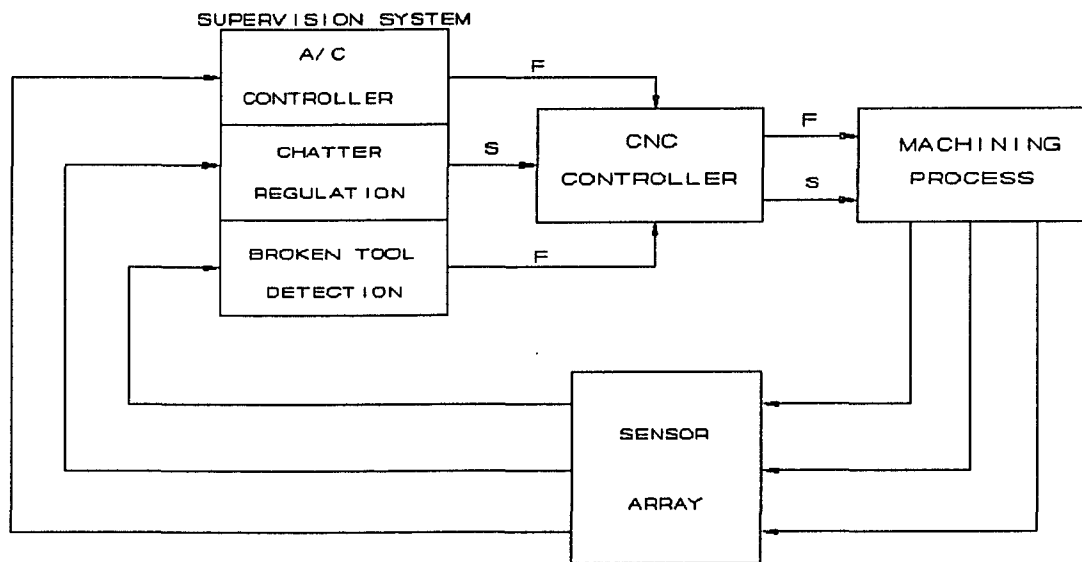


Figure 1.2. Block Diagram of a Machine Tool Supervision System.

the workpiece (obtained from displacement transducers) to monitor both wear and breakage.

Acoustic emission (AE), which refers to high frequency stress waves generated by the rapid release of strain energy within a material, has also been used to monitor tool condition. This signal is obtained by a very high frequency piezoelectric transducer. Balakrishnan et al. (1989) used a combination of AE signals and cutting force signals to monitor tool conditions in turning. However, Tlustý and Tarng (1988, p.46) state that the AE sensor is "sensitive to very sudden events like micro and macro fractures due to chip formation, chip breaking, tool wear and tool breakage . . . [but] . . . problems of robustness are still not satisfactorily solved and the signals are sensitive to variations of cutting data [such as chip breakage, and entry and exit transients]." The required instrumentation is also quite expensive.

The temperature and stress state of the cutting tool can be used as indicators of tool wear in turning, as proposed by Wright et al. (1990b). The difficulty here is in the application of these sensors (thermocouples and strain gages) to rotating tools such as those used in milling and drilling.

Tlustý and Tarng (1988) present a discussion of sensing issues in monitoring tool breakage, and give a detailed comparison of force and vibration signals. They conclude that, although the cutting force gives the best evidence of cutter damage, dynamometers only work well in the low frequency

range. Vibration, as measured by capacitance probes, was shown to give a reliable indication of tool breakage even at high cutting speeds. Their scheme will be installed as part of the supervision system implemented in the present research.

The sensor data from tool condition monitoring can be processed by a variety of Digital Signal Processing (DSP) algorithms. Richter and Spiewak (1989) give an overview of some of the schemes used to extract the characteristic features of tool wear and tool breakage from the sensor signals. Pre-processing by various filtering and averaging schemes, both in the time and frequency domains, and feature extraction by such methods as Auto-Regressive Moving Average (ARMA) models (Lan and Naerheim, 1985) and first and second difference averaging (Altintas et al., 1985; Tlusty and Tarng, 1988) have been used.

Machine tool chatter is a self-regenerative vibration phenomenon that can severely limit the rate of metal removal. Tlusty (1985) gives a detailed analysis of the dynamics involved. He also describes "stability lobes," regions of higher spindle speed where the axial depth of cut in machining can actually be increased without the cut becoming unstable. Figure 1.3 shows a schematic of the mechanism by which machine tool chatter takes place. The phasing,  $\epsilon$ , of the waves left on the machined surface by successive teeth on the cutter can be such that the cutting force,  $F$ , is modulated into a vibration.

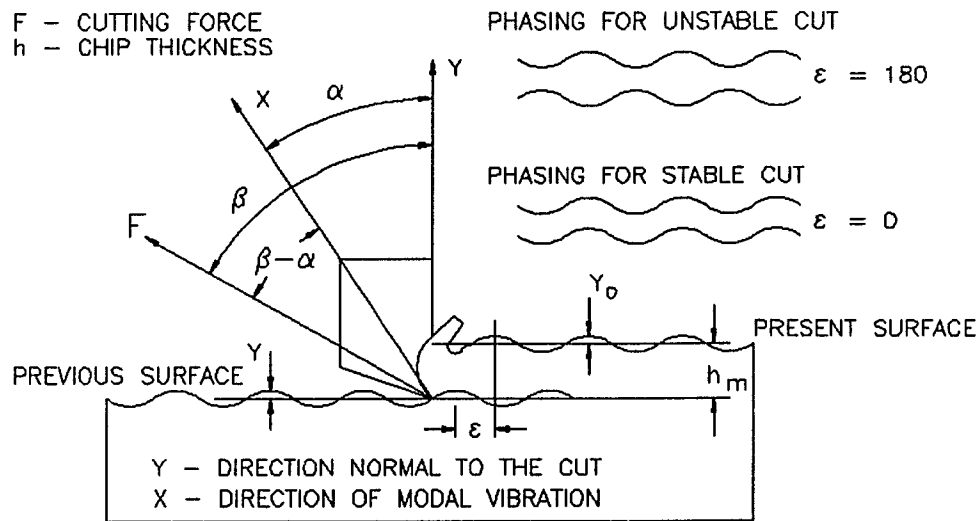


Figure 1.3. Representation of the Chatter Mechanism.

Although a detailed discussion of the formulas that describe chatter is beyond the scope of this work, the expression for the limiting axial depth of a machining cut is of interest:

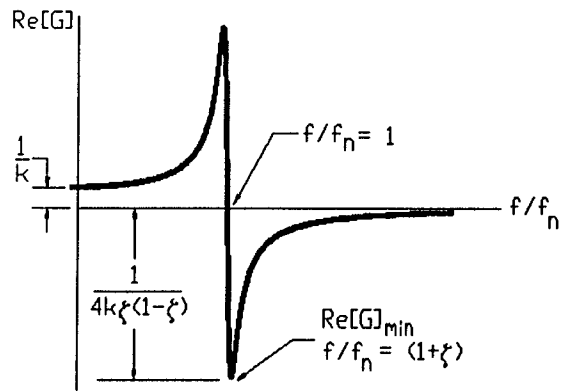
$$b_{lim} = -1 / (2 K_s \text{Re}[G]) \quad (1.1)$$

$K_s$  is the cutting stiffness (specific power) of the workpiece and  $\text{Re}[G]$  is the negative real part of the oriented Transfer Function (i.e. the mode in which the tool-spindle system will vibrate during chatter). The effect of  $b_{lim}$  is to limit the amount of metal that can be removed for a given feed and speed. An effective chatter detection scheme should not only seek to avoid chatter, but also to increase  $b_{lim}$ .

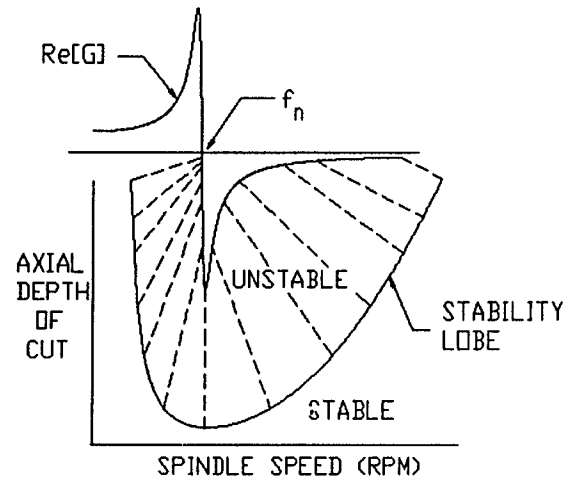
Figure 1.4 shows the real and imaginary parts of a single degree of freedom Transfer Function. It can be seen that the natural frequency,  $f_n$ , the stiffness of the mode,  $k$ , and the damping ratio,  $\zeta$ , can be determined from the plot. Figure 1.5.a shows how the negative part of the Real Transfer Function can be mapped into a stability lobe diagram by varying the spindle speed. If the integer number of waves between subsequent teeth is incremented, a complete lobing diagram is generated. The regions of stability indicated in Figure 1.5.b correspond to the so called "miracle speeds." It has been found that higher metal removal rates can be achieved if the spindle speed can be directed into one of these pockets of stability.

Delio et al. (1990) discuss how chatter has limited the ability of High Speed High Power (HSHP) milling technology to increase the MRR, and describe a system for automatic chatter detection and control using a microphone as the principal sensor. This system will be implemented in the present research and is described in more detail below.

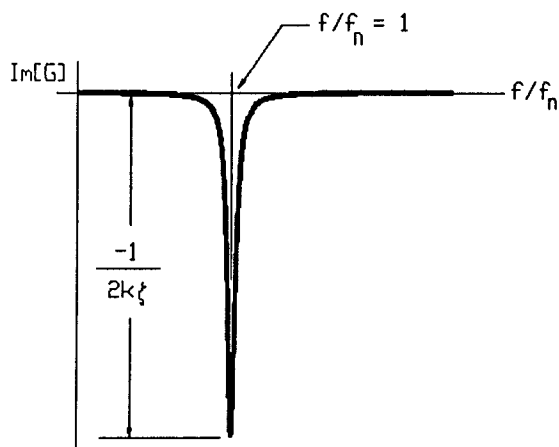
Eman and Wu (1980) studied the on-line identification of chatter in turning from a stochastic approach using a dynamometer to measure the forces on the tool and an accelerometer located on the lathe center. An ARMA model was used to identify chatter. The problem with the stochastic approach, which treats the sensed phenomenon as a pseudo-random "black box," is that no insight is provided into the



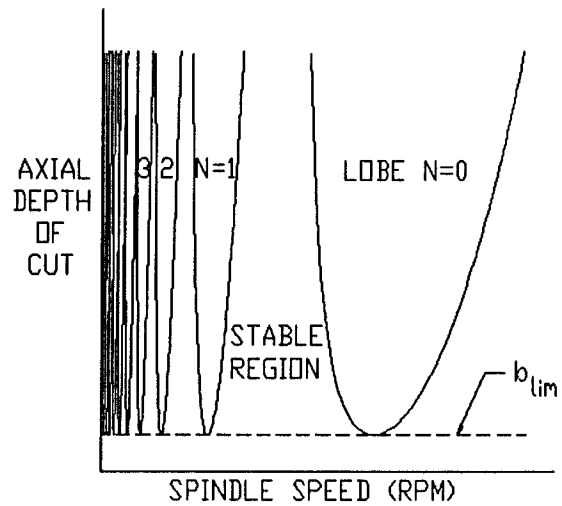
(a)



(a)



(b)



(b)

Figure 1.4. Transfer Function.  
 a) Real Part.  
 b) Imaginary Part.

Figure 1.5. Stability Lobes.  
 a) Lobe Generation.  
 b) Lobing Diagram.

dynamics of the system. Also, often quite high order models are needed to identify the signal, which limits the ability of the system to respond in real time. One merit of the approach, on the other hand, is that it can adapt to changing dynamics of the cutting operation.

Since the dynamics of machine tool chatter are well understood, many researchers have taken a deterministic approach to the problem. In an early work, Weck et al. (1975) used a strain gage torque sensor mounted on the spindle housing to detect chatter. The signal was passed through a slip ring to the control system, which varied the spindle speed to a stable cutting zone determined by vibration measurements taken on the machine tool. The deterministic approach requires that the dynamics of the machine tool be accurately known, but parameters such as the amount of spindle extension, the axial depth of cut, and the location of the workpiece on the machine tool bed (which can vary during a milling operation) can significantly affect the location of stable speeds that can be predicted from stability diagrams.

A deterministic chatter recognition and control system that does not rely on predetermined dynamic characterization of the machine tool is described by Smith and Tlustý (1990) and Smith and Delio (1989). The sound of the milling operation was monitored by a microphone and processed into its frequency spectrum. Using the fact that the chatter frequency usually is close to the frequency of the most flexible mode of the tool-

workpiece system, they found that adjusting the spindle speed so that the fundamental tooth frequency equaled the chatter frequency eliminated chatter by disturbing the regeneration of waviness. Chatter was detected by filtering the spectrum of the tooth frequency and its harmonics and comparing the remaining peak to a threshold value. This scheme will be included in the comprehensive supervision system.

Spindle torque overload detection is used to stop the machining process if the spindle motor is about to stall due to a very heavy chip load. Many machine tool controllers provide the operator information on spindle speed and power by monitoring the motor current and voltage; however, there has been little effort on the part of researchers or machine tool manufacturers in developing a torque overload system.

The spindle motor current has been used as a "sensor" to detect tool breakage (Matsushima et al., 1982), but this approach has not proved practical because the motor's inertia causes it to act like a low pass filter, and only cutter breakage at very low spindle speeds can be detected.

The torque overload detection scheme developed as part of the present research uses an encoder on the spindle motor as a speed sensor. The actual spindle speed is compared to the commanded spindle speed. If the difference is greater than a certain threshold, the cutting process will be stopped before the tool is stalled in the workpiece.



The principal focus of the present research will be the creation of a machine tool supervision system, as suggested by Tlusty and Smith (1989) in their paper on HSHP milling. The discussion, above, of adaptive control schemes touched on the similarity between A/C systems and machine tool supervision, but a review of the literature shows that little has been attempted so far in implementing a comprehensive system that uses multiple sensors to regulate several different aspects of the machining process.

Wright et al. (1990a) describe an open architecture machine tool controller that they have developed as a pedestal for both CAD/CAM research and for the development of sensor-based control strategies. Their system shows a lot of promise, in that its open architecture allows the easy integration of sensors to control the feed and speed of the machine tool. They describe an automatic workpiece locating scheme using a touch-trigger probe. However, most of their efforts so far have been in developing an expert system for workpiece fixturing and tool path verification. The concentration is on integrating the machine tool into an automated factory environment rather than on the real-time monitoring and control of the machining process.

Okafor et al. (1990) used neural networks to integrate multiple sensor signals to estimate surface roughness and bore tolerance in end milling. The cutting force (from a dynamometer), the acceleration of the spindle housing (from an

accelerometer) and the acoustic emission from the spindle housing (from an AE transducer) were the inputs to a network that was trained to correlate these signals to the loss of accuracy caused by tool wear. Although the implementation of this system was focused on monitoring a specific phenomenon, rather than supervision of the machine tool, the approach (and the experimental setup) shows promise as the beginnings of a comprehensive supervision system.

Principe and Yoon (1990) describe an expert system approach to machine tool supervision. This work is part of a cooperative research program between the Machine Tool Laboratory and the Electrical Engineering Department at the University of Florida. In his research, Yoon (1990) developed a Revolution-Oriented Residual Processing Algorithm (RORPA) to detect tool breakage, and integrated the RORPA in a knowledge based supervision system intended to process multi-sensor information from a machine tool. He proposes a hierarchical system where each channel of sensor input is processed for feature extraction by a dedicated DSP chip. The data are then sent to a symbolic processing environment where decisions are made as to the nature of the disturbance, and where action is taken. The machine tool controller would be one component of the distributed system.

So far, Yoon's work has been implemented off line, using data taken from cutting tests performed by Tarng (1988). One objective of the present research program, being pursued by

Walters (1991), is to provide a flexible interface to the machine tool controller so that the expert system can be implemented in real time, and a distributed processing capability can be developed.

### Outline of the Present Research

It can be seen that effective machine tool monitoring and control schemes have been developed to deal with individual disturbances to the cutting process. Some multi-sensor, multi-objective schemes have been developed that could be included in a comprehensive supervision system, but to date no such system has been realized. It is the primary objective of the present research to create such a system.

An external supervisory computer will be interfaced with the machine tool controller. A library of subroutines will be created that permits each of the separate supervision schemes to obtain the necessary sensor data, and to communicate with the machine tool. As discussed in the previous section, the monitoring and control schemes that will be integrated in the supervision system are

1. Adaptive control (Tyler, 1989).
2. Broken tool detection (Tarng, 1988).
3. Chatter recognition and control (Delio, 1989).
4. Spindle torque overload detection.

Each of the existing schemes will be revised as needed to take advantage of the enhanced capabilities for control of the

machine tool. The system will be extremely flexible, and will provide for the future networking of several computers for expert system development and distributed processing.

As an extension of the on-line supervision system, a portable off-line system will also be presented. It can be used for characterization of the dynamics of the machine tool, including data acquisition, Transfer Function measurement, and the spectral analysis of chatter. Information provided by this system could assist manufacturing engineers in identifying dynamic phenomena that disrupt the cutting process.

The original contributions of the present research may be summarized as follows:

1. Create a flexible interface between the machine tool controller and the supervisory computer.
2. Develop a spindle torque overload detection system.
3. Integrate the separate monitoring and control schemes into a comprehensive supervisory system.
4. Create a portable off-line machine evaluation system to complement the on-line machine supervision system.

Chapter 2 of this dissertation is devoted to a detailed description of the White-Sunstrand Series 20 Omnimil machine tool and the Automation Intelligence FlexMate CNC controller. The operation of the FlexMate Motion Co-Processor, which is the heart of the controller, is outlined.

The monitoring and control schemes that comprise the supervision system are described in Chapter 3. The discussion focuses on the algorithms employed.

Chapter 4 concentrates on the supervision system itself. The machine monitoring sensors, and the hardware and software aspects of the interface between the controller and the supervisory computer, are detailed. Then the implementation of the complete supervision system is described.

In Chapter 5, the off-line portable machine evaluation system is described. The features provided in the program PCDATA will be presented and explained.

Chapter 6 contains the results of experimental cuts, made on the Omnimil, that exercise both the on-line supervision system and the off-line machine evaluation system. Machining of aluminum with High Speed Steel end mills and machining cast iron with a carbide insert face mill will be studied.

Chapter 7 contains an assessment of the systems, along with suggestions for future research that could exploit their potential, or extend their capabilities. The need for machine tool controllers to directly support the interfacing of external supervisory computers is discussed.

## CHAPTER 2 THE MACHINE TOOL AND ITS CONTROLLER

A description of the White-Sunstrand Series 20 Omnimil and the Automation Intelligence FlexMate CNC controller is presented. Since the implementation of the supervision system requires close integration with the machine tool controller, the operation of the FlexMate Motion Co-Processor is discussed in detail.

### The White-Sunstrand Series 20 Omnimil

The White-Sunstrand Series 20 Omnimil Machining Center, White-Sunstrand (1983a, 1983b), is a horizontal spindle milling machine with three linear axes, a 360 degree discrete rotary index table and a 30 position automatic tool changer. Figure 2.1 shows a drawing of the machine tool, indicating the standard X, Y and Z linear axis designations.

The three linear axes are driven by high speed armature controlled D.C servo motors, and are positioned by ball screw and nut assemblies. The maximum feed rate for the axes is 400 inches per minute. The ball screws are supported by angular contact ball bearings. The X and Y axis slides ride on rectangular ways by means of recirculating anti-friction

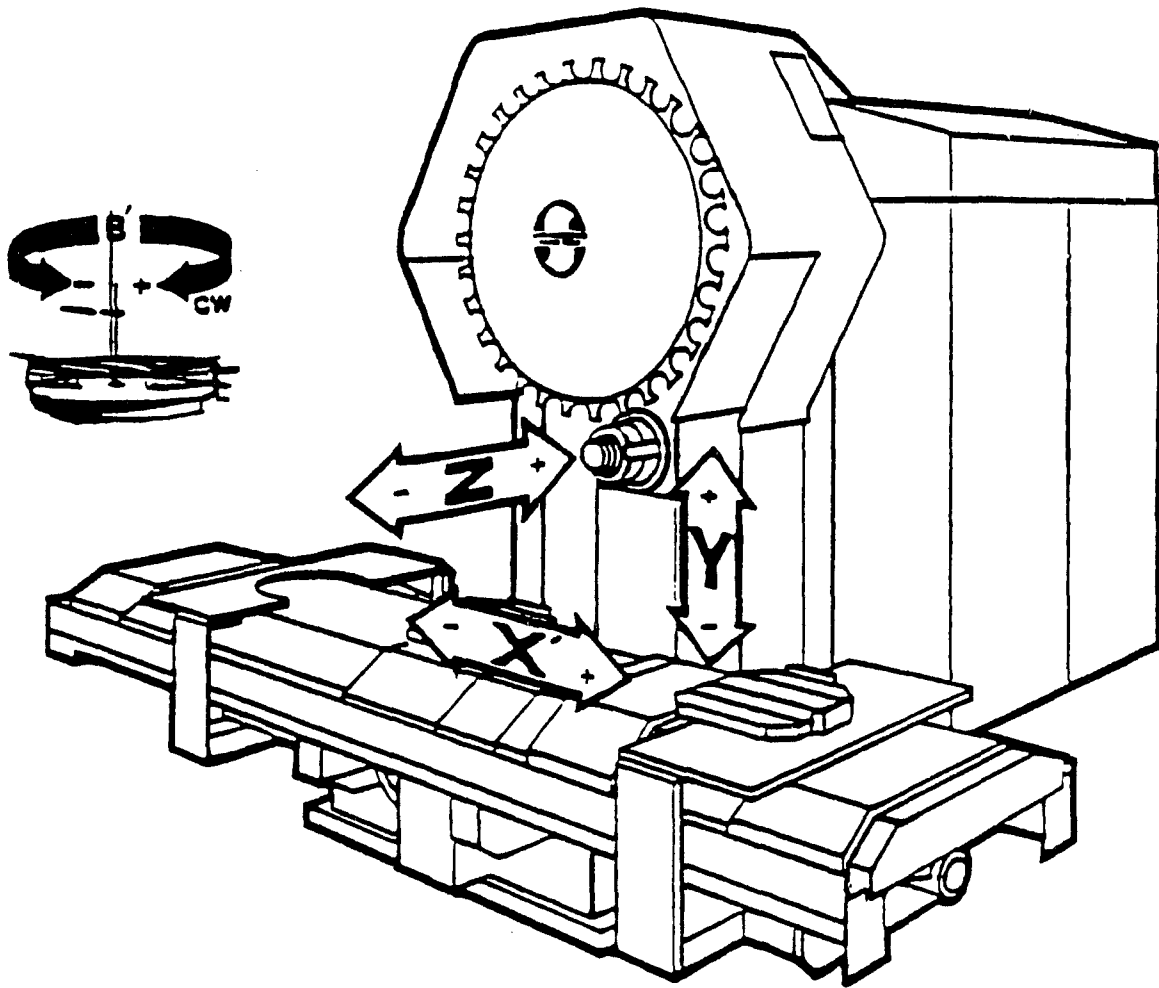


Figure 2.1. The White-Sunstrand Series 20 Omnimil  
(After White-Sunstrand, 1983b).

bearing units. The Z axis is mounted horizontally in the column of the machine and is moved vertically by the Y axis drive. Z axis motion is accomplished by means of a nine inch diameter quill that is supported by two widely spaced linear guide bushings.

The spindle is supported inside the quill at the nose end by two precision angular contact ball bearings and in the rear by one dual row roller bearing. The spindle is driven by a variable speed 25 HP D.C. motor with a maximum speed of 5500 RPM. The spindle accommodates a #50 taper V-flange tool holder that employs hydraulic axial tool retention and a positive drive key.

Each linear axis is equipped with an incremental optical encoder for position feedback to the servo drives. The encoders output 2500 pulses per revolution of input shaft rotation. The encoders are coupled to the drive screws, which have a pitch of 0.5 inches. The effective resolution of the linear positioning systems on each axis is +/- 0.0005 inches.

The D.C servo motors have tachogenerators for velocity feedback and are powered through Silicon Controlled Rectifier (SCR) drive amplifiers. Each axis servo is controlled by an Axis Drive Control (ADC) assembly. The ADC servo boards use the positional error analog voltage ( $E_p$ ), that is output from the CNC controller, and the tachogenerator voltage, to generate the signal that moves the axes. Figure 2.2 shows a schematic diagram of a typical linear axis ADC.



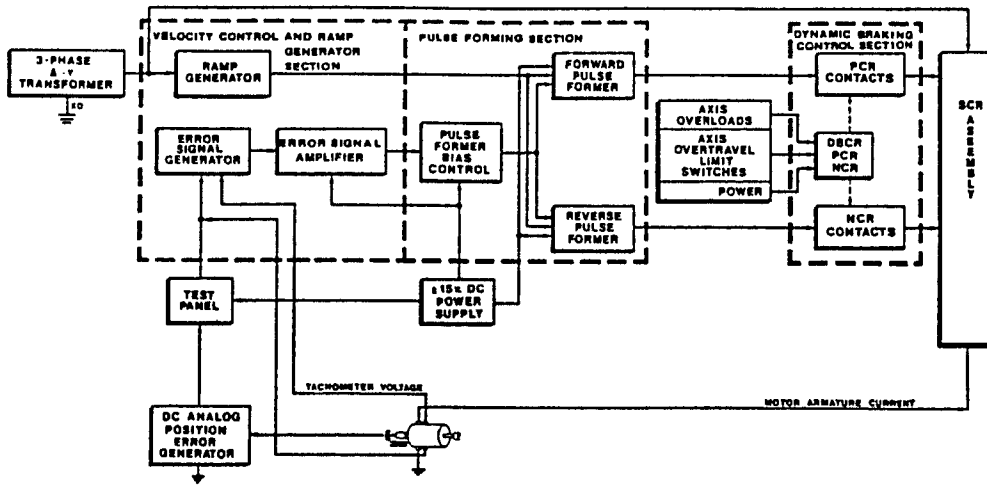


Figure 2.2. Schematic of a Typical ADC Servo Board (After White-Sunstrand, 1983a).

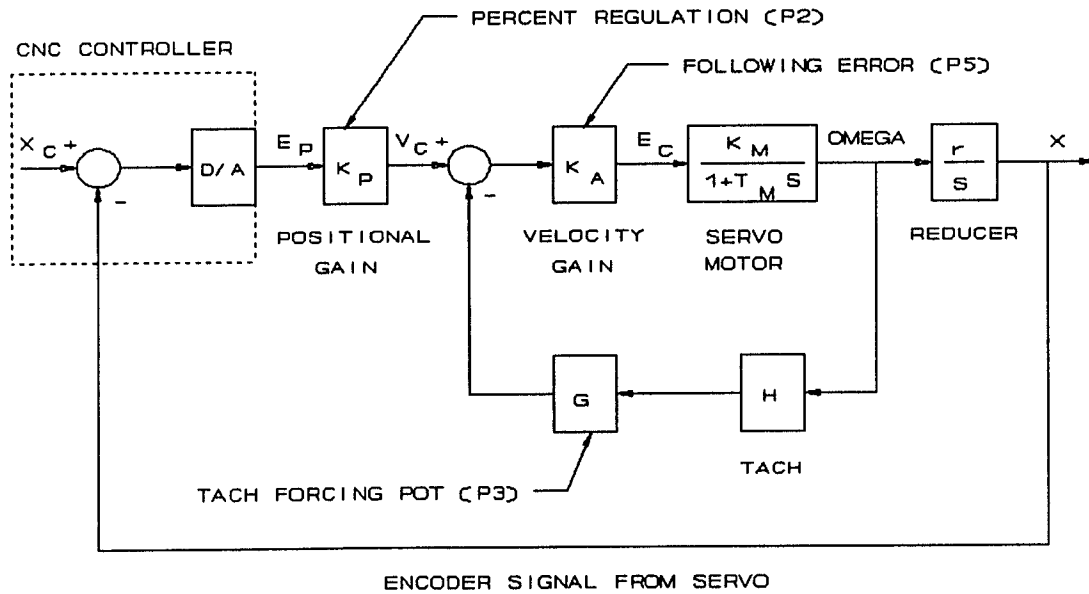


Figure 2.3. Block diagram of a Typical Axis Servo Feedback Loop.

The ADC servo boards provide access for adjusting the positional gain, velocity gain and velocity feedback gain of the X, Y and Z axis servo loops. Also provided is an Axis Test Panel that gives convenient access to the raw positional feedback voltages for each axis. Figure 2.3 shows a classical block diagram of a linear axis positional feedback loop with the ADC gain adjusting pots identified. It can be seen that the ADC servo boards control the velocity loop of the drives, while the CNC closes the positional loop.

#### The Automation Intelligence FlexMate Controller

The Omnimil was originally equipped with a Micro Swinc CNC controller. As part of the present research program, this controller was replaced by a FlexMate CNC controller supplied by Automation Intelligence (AI) in Orlando, Florida. This move was made because the FlexMate provides a flexible and relatively open architecture for customizing the control of the machine tool. As discussed in Chapter 1, it is essential that a sensor-based supervision system have real time access to the machine control parameters of the CNC.

Figure 2.4 shows an outline of the control enclosure in the Omnimil. The ADC servo boards and the Axis Test Panel discussed in the previous section can be seen. The units identified as REMOTE I/O BOARDS, 286 SYSTEM CO-PROCESSOR and MOTION CO-PROCESSOR were installed as part of the AI FlexMate

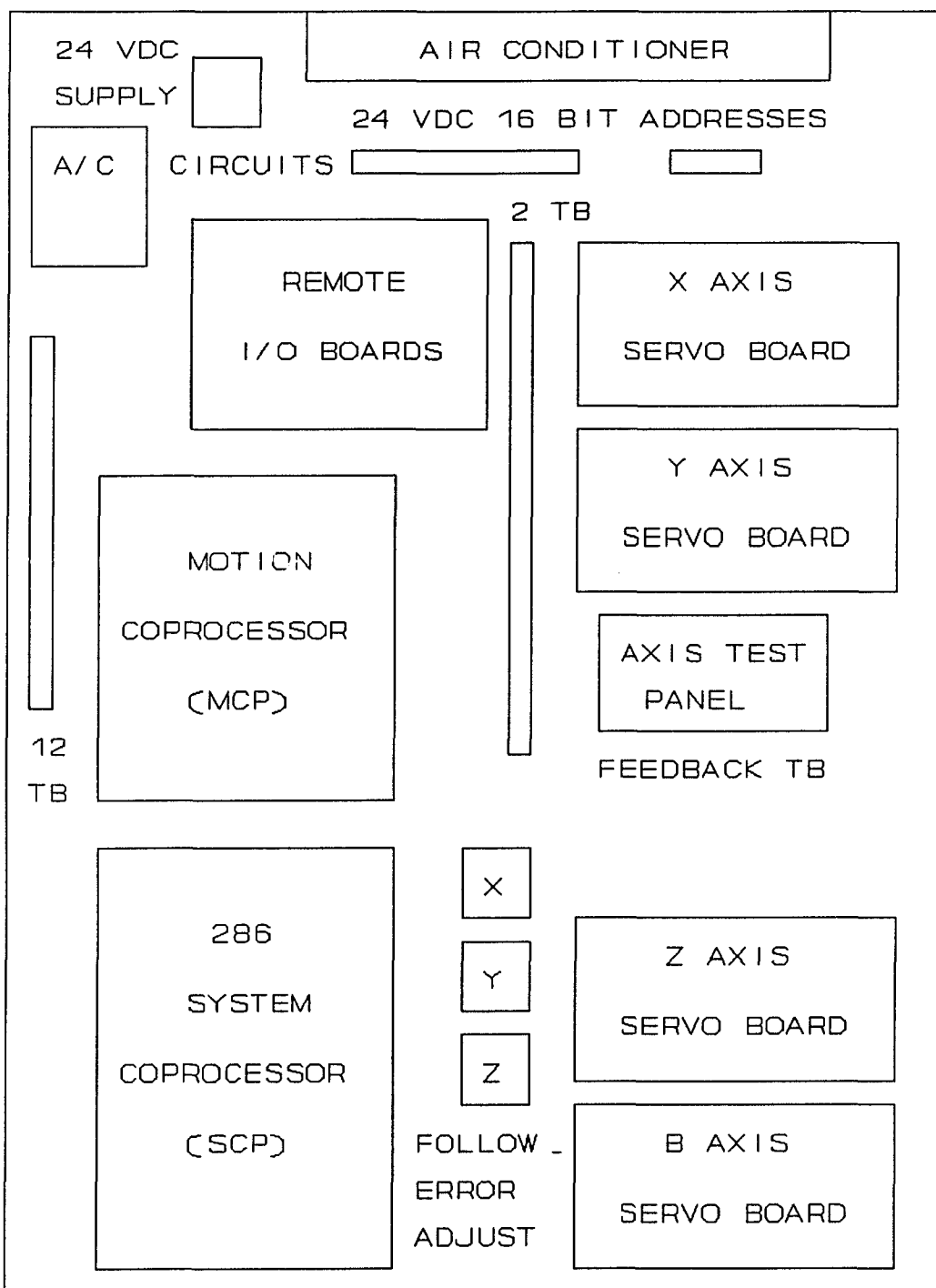


Figure 2.4. The Control Enclosure of the Omnimil Showing the Installation of the FlexMate Controller.

controller, and take the place of the Micro Swinc controller. The "FlexMate Motion Co-Processor Installation and Maintenance Manual" (Automation Intelligence, 1987) describes in detail the function of the individual controller components.

The Remote I/O Boards module is a chassis containing several printed circuit (PC) cards that read inputs from the machine (such as limit switches), and send output signals from the controller to devices on the machine (such as the chip conveyor). The inputs are read as 24 VDC 16 bit logic words, and the outputs are in general 115 VAC signals. The principal function of this module is to step up the working range of the Motion Co-Processor from the 0.0 - 5.0 volt logic range to the voltages required to read and activate devices on the Omnimil.

The System Co-Processor (SCP) is an 80286 8 MHz IBM industrial computer. It has one megabyte of memory, an EGA display adapter, a 30 megabyte hard drive, a 3.5 inch floppy drive and a serial communications adapter with 4 ports. The serial ports COM1 and COM2 are reserved for communication with the Motion Co-Processor. COM3 and COM4 are available for the machine tool user. The SCP is used for loading system software, managing the operator console screen display and loading, editing and sending part programs to the Motion Co-Processor. As will be described in the next section, it is also possible to use the SCP to install application-specific subroutines directly into the Motion Co-Processor, as well as modifying the configuration of the Motion Co-Processor itself.

The SCP uses the DOS 3.3 operating system. However, when the CNC is in operation, a pseudo multi-tasking environment called TASKVIEW is used. This system manages DOS, and divides program execution time into 10 slices (called partitions) of 0.0625 seconds each. The Motion Co-Processor uses every other two partitions, leaving the remainder for use by the machine tool user for specific applications. As far as the present research goes, however, the available processing time in the SCP does not allow the supervision schemes to react quickly enough to sudden phenomena such as tool breakage. For this reason, an external computer must be used, instead of the SCP, to implement the supervisory system.

The Motion Co-Processor (MCP), which communicates with the SCP by means of the two 19.2 kilo baud serial links, contains the PC cards that achieve the actual CNC control of the Omnimil. These cards complete the positional feedback loops by reading the encoders on the axis drives and sending the positional error  $E_p$ , as an analog voltage, to the ADC servo boards. The servo commands are updated each 0.002 seconds. Motion interpolation calculations are also performed here. The interpolations are updated each 0.010 seconds, and in between this time period the interpolations are themselves interpolated. All relevant machine condition information is updated to the CRT screen display, and to the Machine Function Panel on the operator console, from this unit. Figure 2.5 shows a schematic of the complete FlexMate controller.

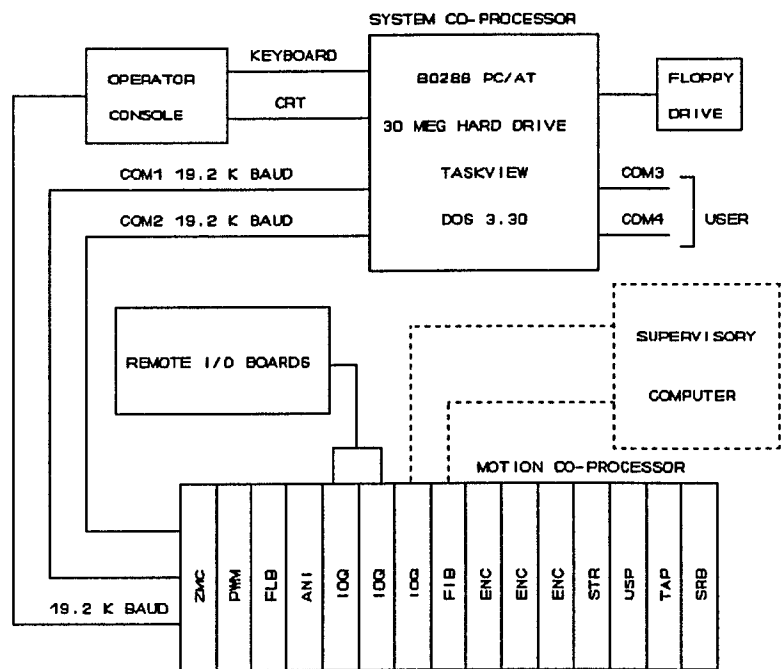


Figure 2.5. A Schematic of the FlexMate CNC Controller.

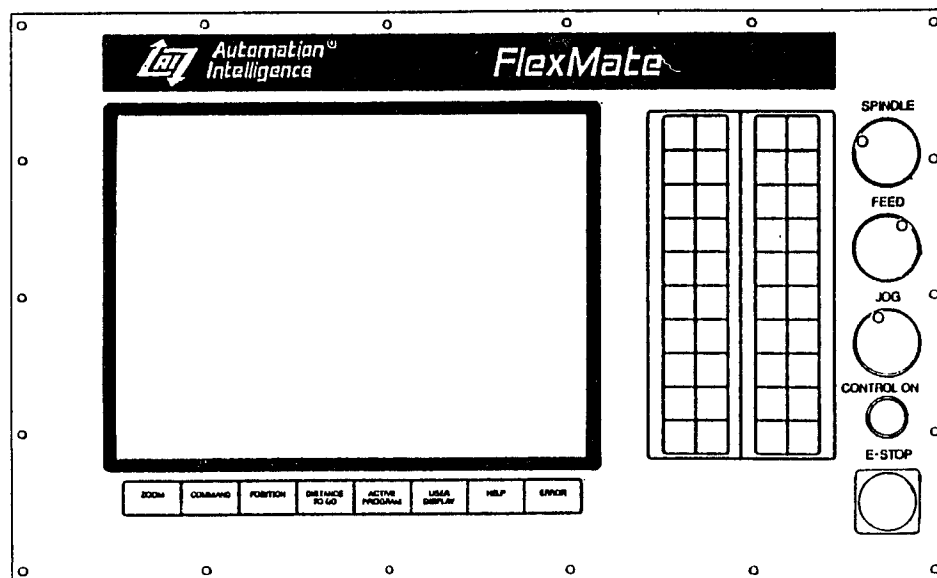


Figure 2.6. The FlexMate Operator Console and Machine Function Panel (After Automation Intelligence, 1989).

Figure 2.6 shows a sketch of the FlexMate operator console. This station includes the Machine Function Panel (MFP), which is to the right of the CRT. The operator monitors the state of the machine tool by selecting various information windows available on the CRT. Part programs can be selected, edited and executed. The MFP provides the operator control of the various machine operations, such as changing the feed rate override. The function of the individual controls on the console is outlined in the "FlexMate Machining Center Operator's Guide" (Automation Intelligence, 1989).

#### The FlexMate Motion Co-Processor

The CNC control of the Omnimil is accomplished by the FlexMate Motion Co-Processor (MCP). Figure 2.7 shows a system schematic of the controller components. The Z-80 Multi Control Card (ZMC) manages communication between the MCP and the System Co-Processor (SCP) using COM1 and COM2. COM1 is used for displays and status on the Operator Control Console, and COM2 is reserved for data and commands. Communication to the Machine Function Panel (MFP) is through an extra 19.2 kilo baud serial link. The ZMC also controls a 3.5 inch floppy drive that is located in the MCP chassis.

The ZMC manages data on the I/O bus that connects the different PC cards inside the MCP chassis, and communicates with the Central Processing Unit (CPU) through an internal

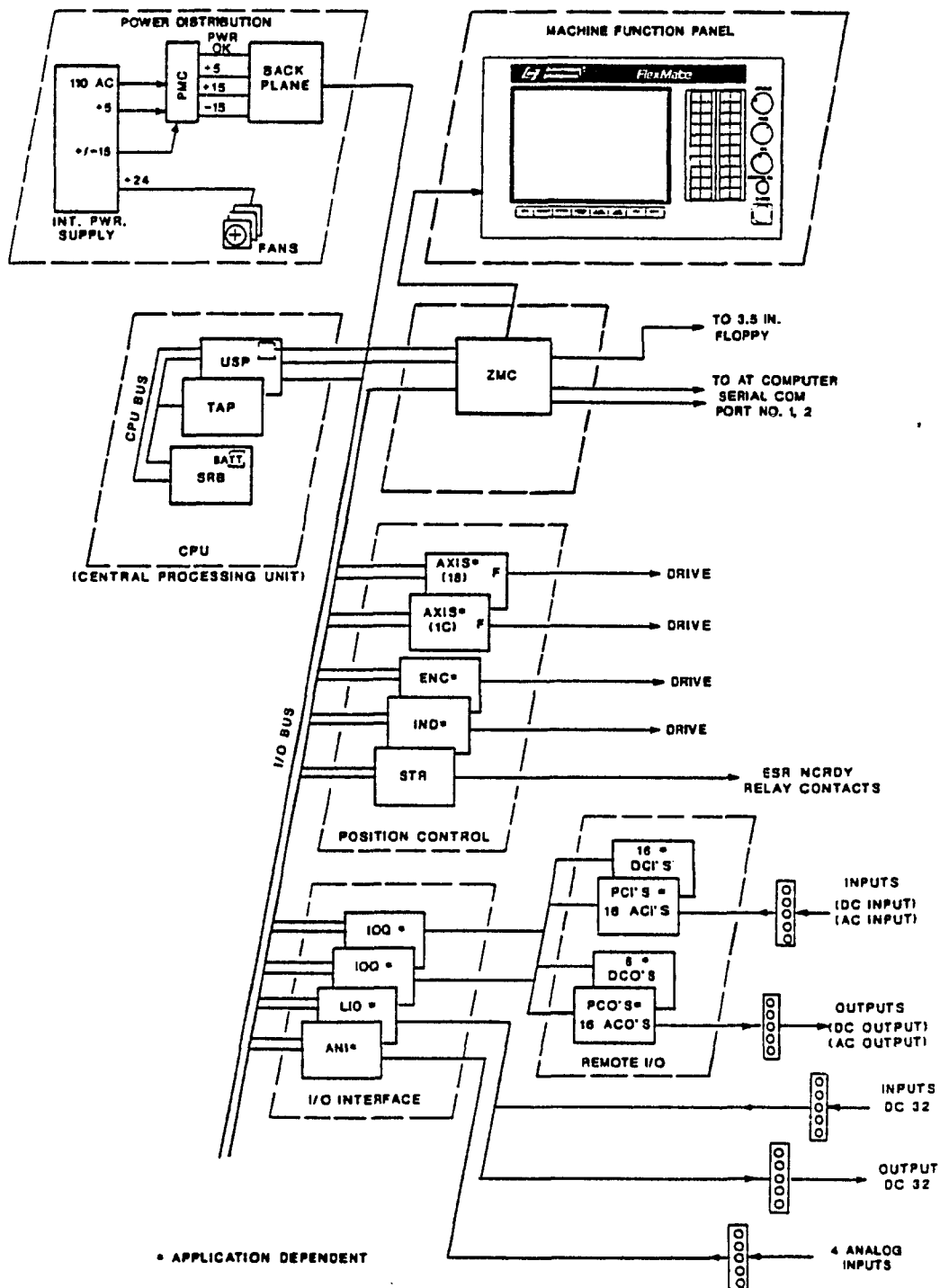


Figure 2.7. Schematic of the Motion Co-Processor (After Automation Intelligence, 1987).



19.2 kilo baud serial link. The CPU, which is the heart of the motion co-processor, consists of three cards. The Static Ram Board (SRB), which has 512 kilo bytes of battery protected memory, stores the executive program and data for operation of the machine tool. The Turbo Arithmetic Processor Card (TAP) is responsible for decoding and execution of the CPU instruction set. The Micro-Support Processor Card (USP) assists the TAP in the execution of the instruction set, and also handles data transfer through the CPU bus and through the serial link to the ZMC. The TAP and USP cards function together to make up the computer that controls the machine tool. The MCP computer is effectively a 16 bit machine with an on-board memory of 256K words.

The System Timing and Relay Card (STR) provides system timing, and the reference voltages for the axes. The board also contains the circuits for remote starting the CPU by the CONTROL ON pushbutton on the operator panel, as well as a "Deadman Circuit" that generates an EMERGENCY STOP if the CPU fails to reset a timer every 0.035 seconds.

The Power Monitor Card (PMC) monitors the +/- 15.0 VDC and + 5.0 VDC power supplies in the MCP. The Floppy Drive Board (FLB) contains the 3.5 inch floppy drive. This drive is used for saving and loading an image of the SRB memory, including the current configuration of the controller. The Analog Input Card (ANI) is a 4 channel, 16 bit, +/- 10.0 VDC Analog-to-Digital converter. Two of its channels are presently

used to read the spindle power output. The other two channels are available.

Feedback from the optical encoder on each axis drive is obtained by a Dual Encoder Interface Card (ENC). There are three ENC's in the MCP. They monitor the X Y and Z axis positions and present the information on the I/O bus to the CPU on demand. The cards are controlled by the CPU, and output the positional error command ( $E_p$ ) computed by the MCP through a 16 bit Digital-to-Analog converter. This signal is sent to the ADC servo boards.

The Input/Output Converter Driver Card (IOQ) is used to interface up to 64 bits of management and operational data between the CPU and the REMOTE I/O card cage that is interfaced to the machine tool. There are two IOQ boards used in the FlexMate for MCP control of the Omnimil, operating in the 0.0 - 5.0 volt TTL logic range. The remote I/O card cage contains a number of Point Contact I/O Cards (PCI/PCO). These cards provide the interface between the TTL logic of the MCP and the 24 VDC and 115 VAC signals used in the machine tool.

The interface between the MCP and the external computer containing the supervision system uses a third IOQ card. This card is used to establish a 32 bit parallel interface for receiving machine tool status information from the MCP and requesting machine functions such as FEED HOLD from the MCP. The ability to interrupt the MCP so that data can be transferred through the IOQ interface is provided by the Fast

Input Board (FIB). The FIB is an 8 bit maskable interrupt generator that is software compatible to the IOQ. It can interrupt the MCP within 0.002 seconds, and the interrupt can be pointed at a designated application program in the MCP.

There are two event areas within the MCP, called OEM\_MAIN and OEM\_SYNC, where user-supplied Logic Control Language (LCL) programs can be activated in the CPU when predetermined bit patterns appear on the FIB. The supervision system interface software utilizes the OEM\_MAIN event area to enable communication between the remote supervision computer and the MCP. The OEM-SYNC area is used to implement a special fast stopping routine that is needed by the supervision schemes.

The MCP is an interrupt driven system. There are 32 tasks ranging from the lowest priority, level &H0, to the highest priority, level &H1F. When an interrupt occurs, the task scheduler is invoked. It looks for the highest priority task that is both active and able, and executes it. Table 2.1 lists the FlexMate task structure as installed on the Omnihil.

MAIN (level &H3) is a low priority event area that runs continually in the background. SYNC (level &H1D) is a high priority foreground task that interrupts MAIN every 0.010 seconds. LCL code in the SYNC area must execute to completion within the 0.010 seconds. Code in the MAIN area, however, is continued from the point where it was suspended after each 0.010 second time out. The MCP memory available for OEM\_SYNC code is 2 kilo bytes. This area is intended for programs that

are short and which must run to completion within 0.010 seconds. The memory available for OEM\_MAIN code is 64 kilo bytes, and the execution time can be as long as the user requires.

Table 2.1. Flexmate Interrupt Task Structure

---

1F	BUS TIMEOUT	F	RAM DISK PORT 1
1E	(NOT USED)	E	RAM DISK PORT
1D	SYNC	D	ZMC READ HANDLER #1
1C	DNC HANDLER	C	ZMC WRITE HANDLER #1
1B	ZSC READ HANDLER #2	B	ZMC MODEM HANDLER #1
1A	ZSC WRITE HANDLER #2	A	MFP OUTPUT
19	ZSC MODEM HANDLER #2	9	OPERATOR PANEL KEYBOARD
18	RAM DISK PORT 3	8	OPERATOR PANEL OUTPUT
17	RAM DISK PORT 0	7	(NOT USED)
16	DISK1 PORT1 HANDLER	6	(NOT USED)
15	DISK1 PORT0 HANDLER	5	OPERATOR PANEL INPUT
14	DISK2 PORT1 HANDLER	4	ZMC READ TASK
13	DISK2 PORT0 HANDLER	3	MAIN
12	DISK READ TASK	2	PLC PUNCH/LOAD
11	CONFIGURATION PACKAGE	1	CRT APPLICATION
10	FMS TASK	0	POWER ON SEQUENCE

---

The FlexMate provides a program development environment for writing LCL programs. The language is a generic version of the C language. The source code is compiled, assembled and linked into the proprietary executable code that will run on the MCP computer. A library of "Window Functions" is supplied that gives the programmer access to most of the critical parameters of the CNC. The interface and the fast stopping routine make use of several of these functions.

The memory of the MCP is divided into several discrete regions. High memory is reserved for the MCP executive program that actually performs the machine motion control, including interpolations. Also, programs that run in OEM\_SYNC and OEM\_MAIN are assigned space in this area when they are loaded into the MCP. The System Data Table, starting at address &H2C00, consists of variables that can be read and changed by an LCL program. The System Constant Table, starting at address &H1000, contains flags for the interrupt logic as well as the definitions of the FlexMate's I/O space and the motion control tables. This area may be considered the heart of the MCP. The Operating System, which starts at &H00FF, contains the task and I/O scheduler and the interrupt vectors. The Zero Page area, which begins at &H0000, contains the system pointers and counters.

The flags and tables that show the status of the drive and motion programs in the MCP are collected into a master table in the System Constant Table called "MOSTAT" at address &H17DD. Figure 2.8 shows a simplified block diagram of how the positional feedback loop is closed inside the MCP. The variable names in the block diagram refer to per-axis tables that are contained in MOSTAT. The motion interpolator outputs an incremental position command to the table XREF (the absolute axis reference is held in a table called XCOMM). Feedback from the axis encoders is processed into engineering units and held in the XFBK table. XERR is the difference



## CHAPTER 3 THE MACHINE MONITORING AND CONTROL SCHEMES

Before the implementation of the supervision system is presented, it is important to describe the individual machine monitoring and control schemes that will be employed. With the exception of the Spindle Torque Overload Detection System, which was developed as part of the present research, the schemes are the result of previous research programs conducted at the Machine Tool Laboratory of the University of Florida. A special fast stopping routine that is used by all the schemes is also presented.

### Adaptive Control System

The Adaptive Control (A/C) system was developed by Tyler (1989). One purpose of the system is to allow the tool to be moved at rapid speeds when there is no metal cutting. When the tool encounters the workpiece, the impact is detected and the axis feed is stopped by a fast stopping routine (described below). After the tool feed has stopped, the servo loop is reconnected and the adaptive control loop begins regulating the cutting feedrate.

The adaptive control loop increases the feed rate during the actual cutting until either the specified feed rate or the vibration limit is reached. The fast stop is also called if the vibration exceeds the specified threshold in the adaptive control loop. The adaptive control loop runs constantly during the cut. If the vibration falls below a lower threshold, indicating a non-cutting condition, the program returns to the fast transient mode. Figure 3.1 shows a flow chart of the Adaptive Control system.

The hardware of the A/C system will be described in detail in Chapter 4. The system was originally implemented using a Kistler model 9067 table dynamometer to measure the cutting force, but for the present research a displacement signal was used instead. This change of sensors was made because the dynamometer has been shown to distort frequencies above about 70 Hz in the machine X axis direction, and above about 200 Hz in the machine Y axis direction (Smith, 1985; Tarng, 1986; Tyler 1989). A sensor ring containing four inductance probes (two for the X axis and two for the Y axis) has been fitted to the spindle housing. A signal conditioning system presents the X and Y axis vibration signals to a data acquisition board in the supervision computer.

Originally, data acquisition for the A/C system was externally triggered and timed by the pulses from a variable reluctance magnetic pickup that read the passage of the teeth of a 72 tooth gear attached to the spindle. For the



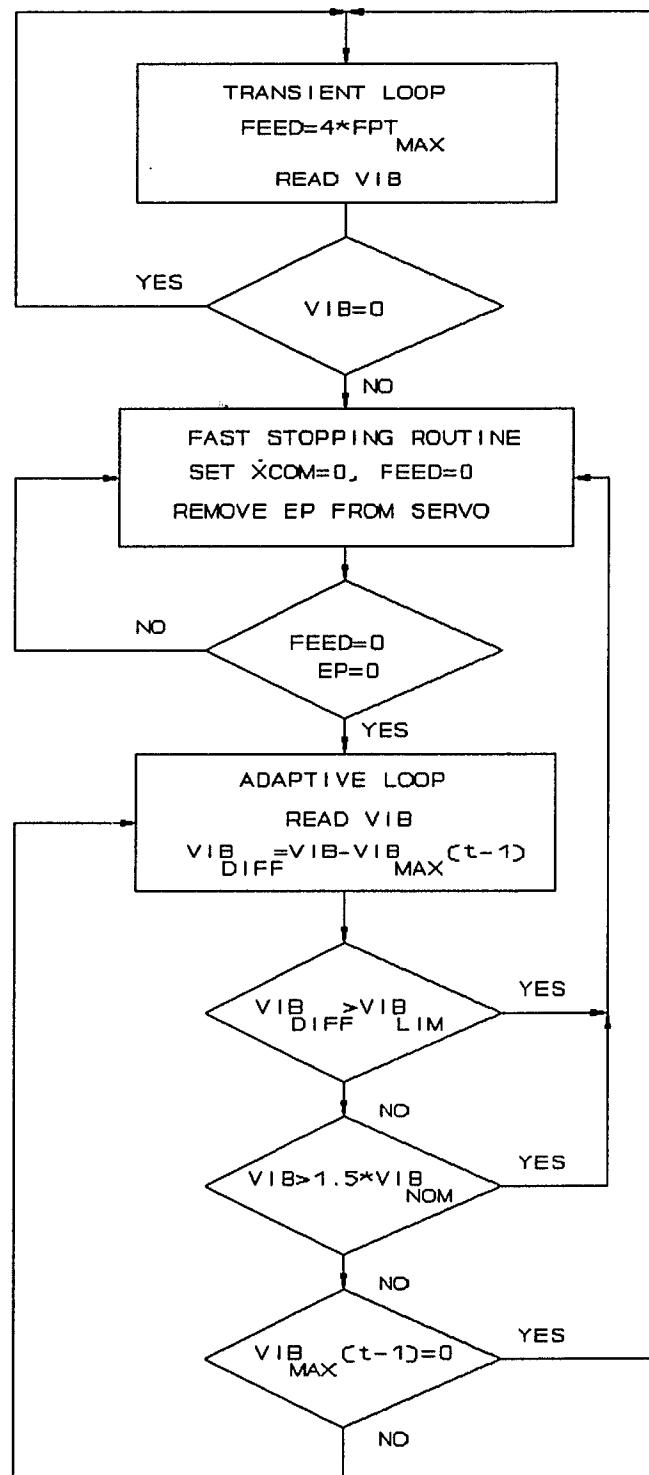


Figure 3.1. Flow Chart of the Adaptive Control Algorithm (After Tyler, 1989).

implementation of the supervision system, a 240 line encoder attached to the spindle shaft was used instead of the pickup and gear. This change was made because of the improved resolution in synchronizing the data sampling that is possible from the encoder, as well as the advantage of moving the sensor away from the cutting process.

Control of the axis feedrates, based on the vibration of the cutting process, is accomplished by varying the percentage of feedrate override using one of the supervision subroutines created as part of the present research. The redesign of the A/C system to use the new sensors, and the supervision subroutines, is described in Chapter 6.

#### The Fast Stopping Routine

The fast stopping routine used in the A/C system was originally implemented as a hard-wired interface to the CNC controller. A relay was used to break the line carrying the following error command ( $E_p$ ) to the servos. After the drive had stopped, a small voltage was used to remove the remaining following error, and the control loop was reconnected. As part of the present research, the routine has been redesigned to take advantage of the interface to the MCP.

Figure 3.2 shows a flow chart of the Fast Stopping Routine. The algorithm uses the interface software `iface.c`, installed in the `OEM_MAIN` area of the MCP, and a specialized

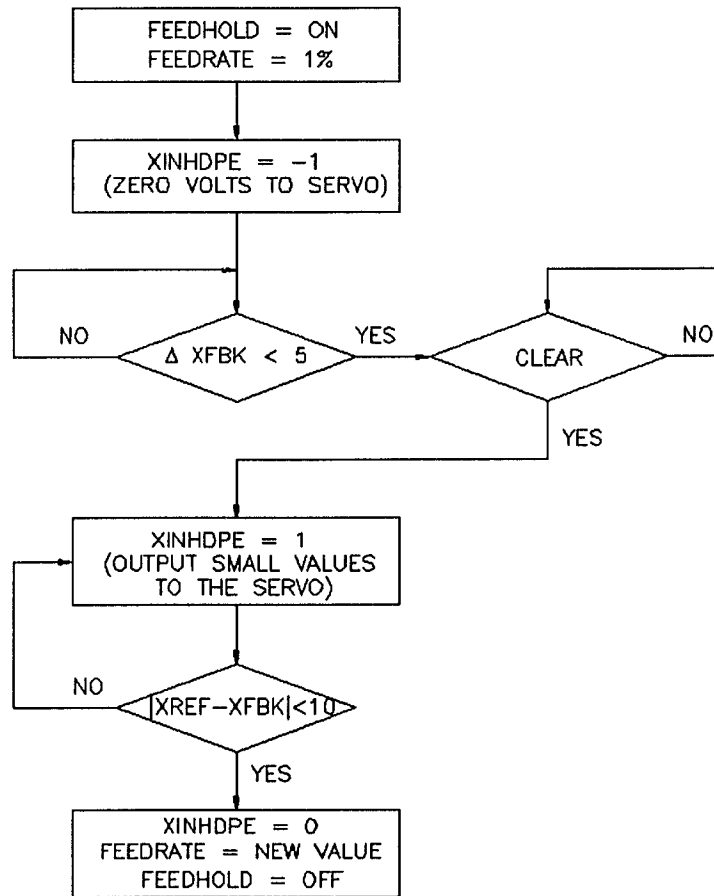


Figure 3.2. Flow Chart of the Fast Stop Algorithm.

routine, `stop.c`, installed in `OEM_SYNC`. The programs are listed in Appendix A, and complete details of the routines are described in the report by Wells (1991a).

When the `FastStop` subroutine is called, the function called `iface_interrupt`, running in `OEM_MAIN`, stops interpolation by issuing an internal feedhold to the MCP. The X and Y axis drives are disconnected from the CNC, and zero volts are written to the servos, by setting the `XINHDPENeg1` table

flag to -1. When the fast\_stop\_flag is set equal to 1, the code in OEM\_SYNC takes over on the next SYNC pass. The routine waits for the axes to slow down by monitoring the change in the feedback position using the XFBK table. The feedhold is held on until the ClearFastStop subroutine is executed.

When the fast stop is cleared, the XINHDPPE table flag is set equal to 1, which enables the axis D/A converters for output. A small voltage is output to move the servos until the CNC positional error (which is the difference between the XREF and XFBK tables) is within +/- 0.001 inches. When the axes are in position, XINHDPPE is set equal to 0, which reconnects the servos to the CNC position command generator.

The fast stopping routine is useful for all the machine supervision schemes, since it allows the feed to be stopped much faster than is possible by a conventional feedhold. In Chapter 6, the performance of the routine will be evaluated as it is exercised as part of the supervision system.

### Tool Breakage Detection System

The Tool Breakage Recognition system (Tarnng, 1986, 1988) is designed to stop the axis feeds immediately when tool breakage occurs. A flow chart of the algorithm is shown in Figure 3.3. The data acquisition is triggered and timed externally from the spindle encoder, and the vibration of the tool is sensed by inductance probes. It can be seen that the

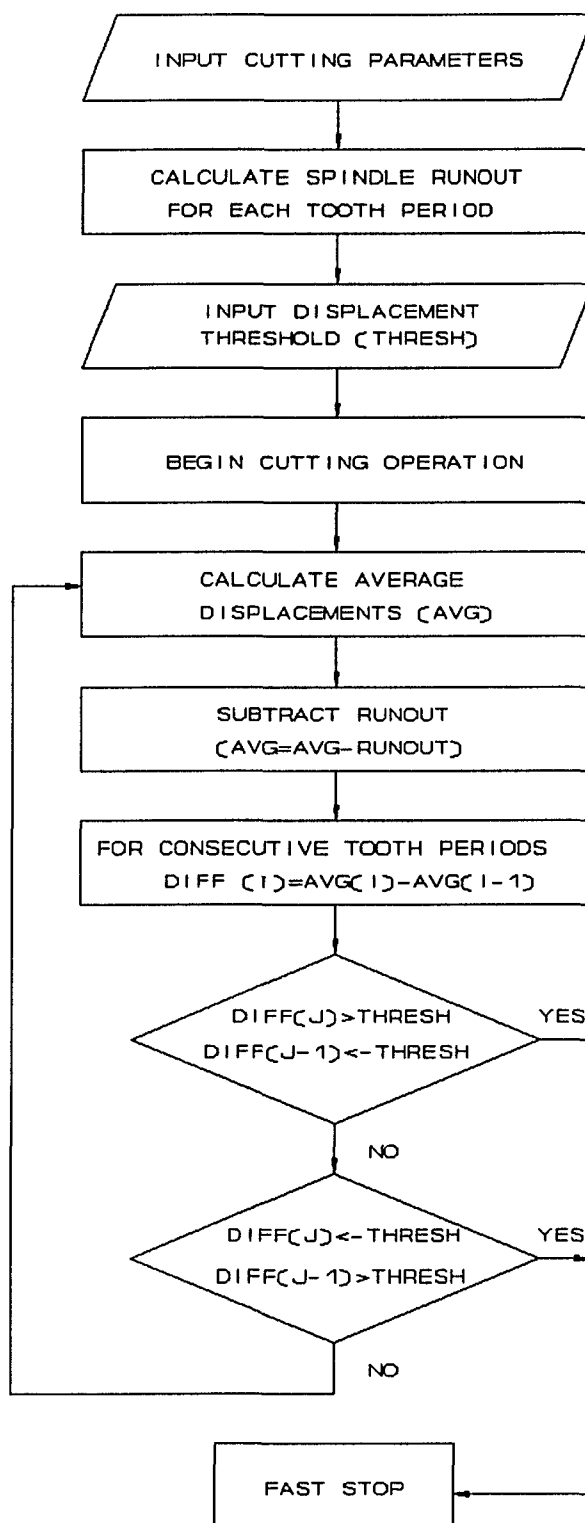


Figure 3.3. Flow Chart of the Broken Tool Detection Algorithm (After Tarng, 1988).

same sensor signals used in the A/C system are used for the tool breakage system. If tool breakage is detected, the program calls the supervision subroutine that issues a fast stop command to the CNC controller.

The scheme has two main parts, spindle runout determination and tool breakage detection. The spindle runout section samples spindle vibration in synchronization with the tool rotation while the tool is not cutting. The samples are taken following a once-per-revolution trigger signal, and the displacement values are averaged for each tooth period.

The tool breakage detection portion of the program samples and averages the spindle vibration in the same way as the runout portion. The runout values are subtracted from the new values to separate the cutting deflections from the runout. The difference between consecutive average deflection values are then taken as an indication of how the forces on the tool are changing from tooth period to tooth period. For an undamaged cutter, the difference values will remain between upper and lower threshold limits. When a cutter is damaged, however, the consecutive difference values will exceed both the positive and negative threshold limits and the algorithm issues a fast stop command.

As a broken tooth enters the cut, its chip load is smaller than for the unbroken teeth, which results in a sharp change in the spindle deflection. When the next unbroken tooth enters the cut, it has an increased chip load which results in

an opposite change in the spindle deflection. Figure 3.4 shows a detail of the first difference of the vibration signal from an eight tooth 4.25 inch diameter face mill cutting cast iron with one damaged insert. The signature of tool breakage can clearly be seen. Transients such as entry to and exit from the workpiece, hitting a hard spot in the material, or milling over slots have been shown by Tarng (1988) not to produce this effect and will not be mistaken as tool breakage.

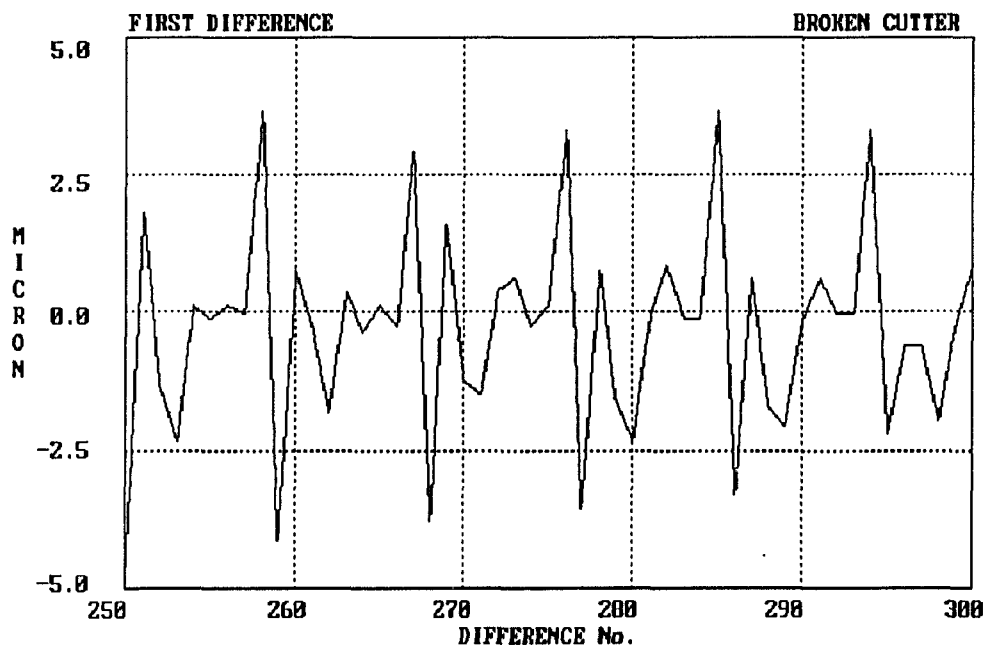


Figure 3.4. Detail of the First Difference of the Vibration Signal from a Damaged Face Mill.

Adaptive thresholding for the system is currently being studied in the Machine Tool Laboratory (Vierck, 1991). Adaptive signal processing schemes and digital filtering are also being studied as ways of identifying the characteristic

tool breakage signal. The distributed supervision system being developed by Walters (1991) also includes a real time application of the RORPA algorithm of Yoon (1990).

### Chatter Recognition and Control System

Smith (1985, 1987) studied forced vibrations and chatter in high speed milling, and developed a strategy for chatter regulation by spindle speed selection. The real time chatter recognition and control system was created by Delio (1989), and was implemented on a vertical milling machine, manufactured by Lamb, using a 9000 RPM Setco spindle. The CNC controller was a General Electric Mark Century 2000. The scheme was further developed by Keyvanmanesh (1990) to take advantage of the 25000 RPM range of a Setco experimental high speed spindle. A simplified flow chart of the algorithm is presented in Figure 3.5.

The system works by sampling the cutting noise from an audio microphone near the machine. Two microphones could be used for directionalization if the system is affected by noise from other machines. A Digital Signal Processor (DSP) performs a Fast Fourier Transform (FFT) on the microphone signal in real time. The supervision computer then processes the FFT signal, filtering the tooth frequency and its harmonics, and determines the frequency of chatter if it exists. The same fast stopping strategy used in the A/C system is used to stop



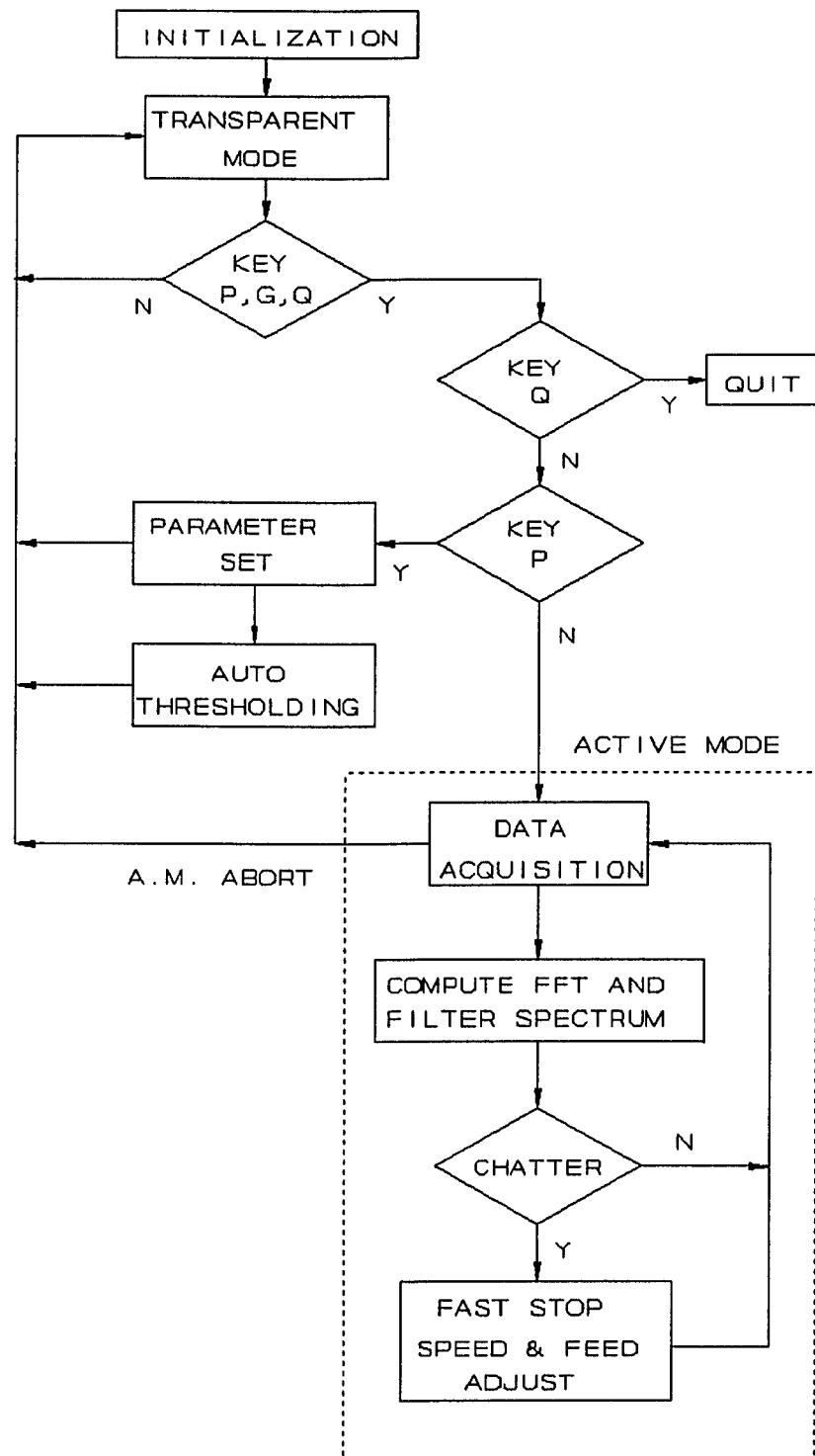


Figure 3.5. Flow Chart of the Chatter Recognition and Control Algorithm.

the feed. This minimizes the amount of bad surface that is generated. Then, based on the relationship between chatter and spindle speed described by Smith (1987), the algorithm commands a new spindle speed to the machine tool. The feedrate is also increased in order to maintain the same feed per tooth on the cutter.

In order to detect chatter it is essential to be able to reject the runout and tooth frequency harmonics that are seen in the spectrum of the cutting signal. For this reason, the spindle speed must be accurately known in real time. Also, the means of controlling chatter relies on setting the spindle to a speed such that the tooth frequency of the tool equals the chatter frequency that was detected. A digital tachometer that can resolve speeds to +/- 2 RPM is used in the supervision system to sense the spindle speed on the Omnimil.

Figure 3.6 shows an example of the spectrum of a stable machining cut. The data was sampled by the off-line machine evaluation program described in Chapter 5. The tooth frequency, which is itself an harmonic of the spindle frequency, can be clearly seen. Figure 3.7 shows the spectrum of an unstable cut made with the same tool. The chatter, which has a frequency of 3623 Hz, was caused by increasing the axial depth of cut beyond the limiting value for stability,  $b_{lim}$ . It can be seen that the chatter peak and its frequency could now be clearly identified in the spectrum if the tooth frequency and its harmonics were filtered.

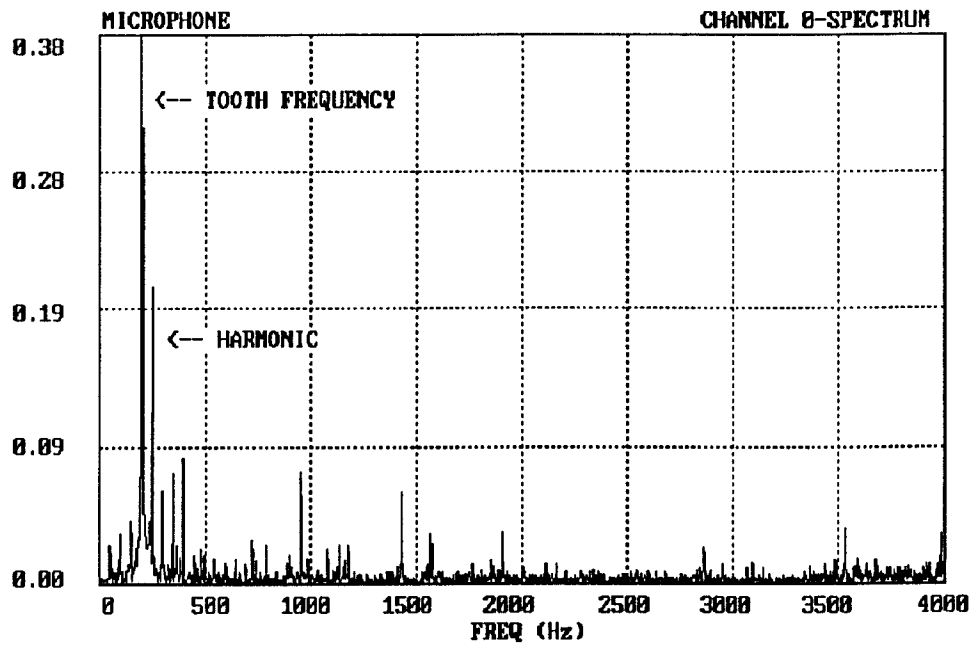


Figure 3.6. Spectrum of a Stable Machining Cut.

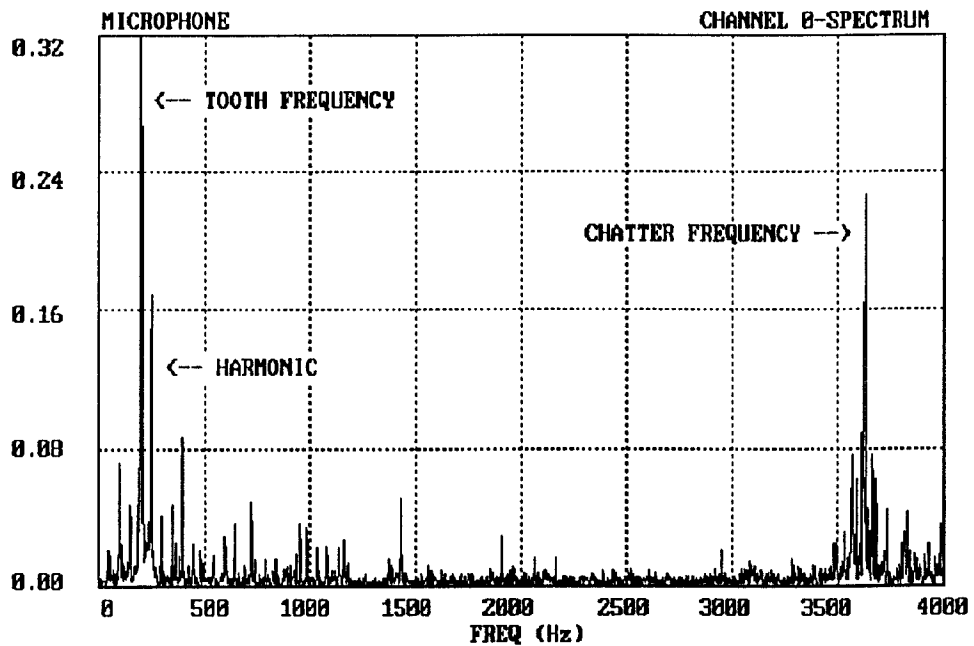


Figure 3.7. Spectrum of an Unstable Machining Cut.

### Spindle Torque Overload Detection System

As part of the present research, a monitoring scheme was developed that senses when the spindle motor is about to stall. This problem occurs when the tool is cutting too heavy a chip load, and can damage both the tool and the workpiece. A flow chart for the algorithm is shown in Figure 3.8, and a listing of the system software is presented in Appendix B.

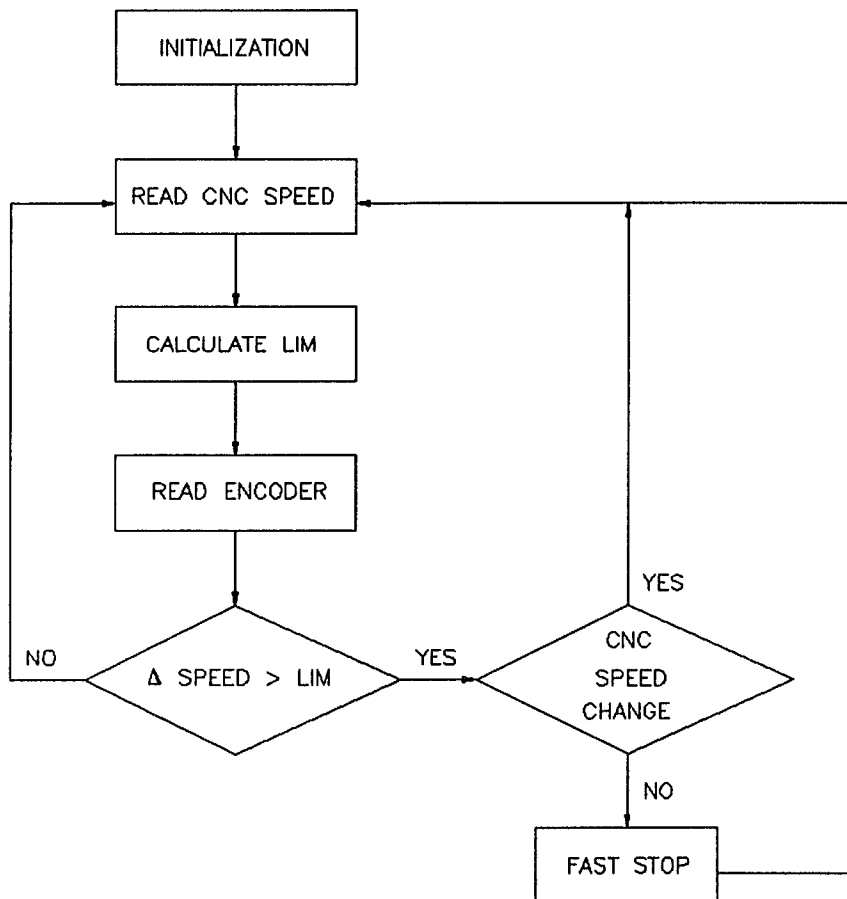


Figure 3.8. Flow Chart of the Spindle Torque Overload Detection Algorithm.

The output torque of an armature controlled DC motor is produced by the current in the armature. This torque accelerates the drive inertia, and must overcome the external load torque and the losses due to windage and friction. This relationship may be expressed as

$$T = K_t i = J\alpha + B\omega + T_l \quad (3.1)$$

where  $T$  is the motor torque,  $K_t$  is the motor torque constant,  $i$  is the armature current,  $J$  is the motor and drive inertia,  $\alpha$  is the motor acceleration,  $B$  is the loss coefficient,  $\omega$  is the motor speed, and  $T_l$  is the load torque. Usually the windage and friction losses are small compared to the other parameters and may be neglected. Rewriting the above equation with the loss term deleted and solving for acceleration gives

$$\alpha = (T - T_l) / J \quad (3.2)$$

This expression shows that monitoring the speed change gives a direct indication of the spindle motor torque.

The scheme works by reading the CNC spindle speed in real time using one of the supervision subroutines. The speed threshold is input as a percentage of the CNC commanded speed. It was determined after watching the speed variation during various machining cuts that a 95 percent limit will not cause any false triggers of the system.

The actual spindle speed is read and compared to the limit speed, and if the speed change is beyond the threshold then a fast stop command is issued to the Motion Co-Processor. When the operator clears the fast stop, the system returns to its monitoring mode. The scheme is designed to track legitimate speed changes commanded from CNC code blocks, the operator console or from the other supervision schemes, so that only a speed change caused by an impending stall will trigger the fast stop.

The spindle motor on the Omnimil has tachogenerator feedback, but the high AC component of this voltage makes it unsuitable for sensing the spindle speed. In order to implement a tachometer in the supervisory computer, the pulse train from the 240 line spindle encoder is passed through a divide-by-240 counter, producing one pulse per revolution. This signal is also used to trigger data acquisition for the other supervision schemes, as will be described in Chapter 4.

Since the standard timer available in the supervisory computer can only resolve 0.052 second intervals, a strategy was devised to strobe the printer port with the once-per-revolution signal after it was converted from a TTL pulse train to an impulse train that could be read by the port. By redirecting the interrupts from the DOS clock and the LPT1 printer port, the signal is processed into a digital tachometer that can resolve speeds to +/- 2 RPM, and which has an effective operating range up to 5000 RPM. John Frost of

Manufacturing Laboratories contributed the interrupt code and circuits necessary to implement the tachometer.

In Chapter 4 an inventory of the sensors and data acquisition hardware needed for the supervision system will show that the individual schemes share most of the same sensor inputs. It will also be shown that only a relatively small number of supervision subroutines are needed to provide the necessary control of the machine tool.

## CHAPTER 4 THE ON-LINE MACHINE SUPERVISION SYSTEM

The actual implementation of the on-line supervision system will be described in this chapter. The sensor signals required for the system will be discussed, along with the data acquisition hardware used in the supervision computer. The hardware layout of the interface between the computer and the FlexMate controller will be described, as well as the interface protocol used between the CNC controller and the supervisory computer. The supervision subroutines will then be presented, along with a description of overall system.

### An Inventory of the Sensors

Each machine supervision scheme described in Chapter 3 needs its own sensor input. The schemes have been designed, however, to make use of shared sensor data. The adaptive control system and the tool breakage system both need X and Y axis vibration signals, as well as the spindle speed for data sampling. The torque overload system needs only the spindle speed. The chatter control system needs a microphone signal and the spindle speed. Figure 4.1 shows the placement of the sensors, and their connection to the supervisory computer.



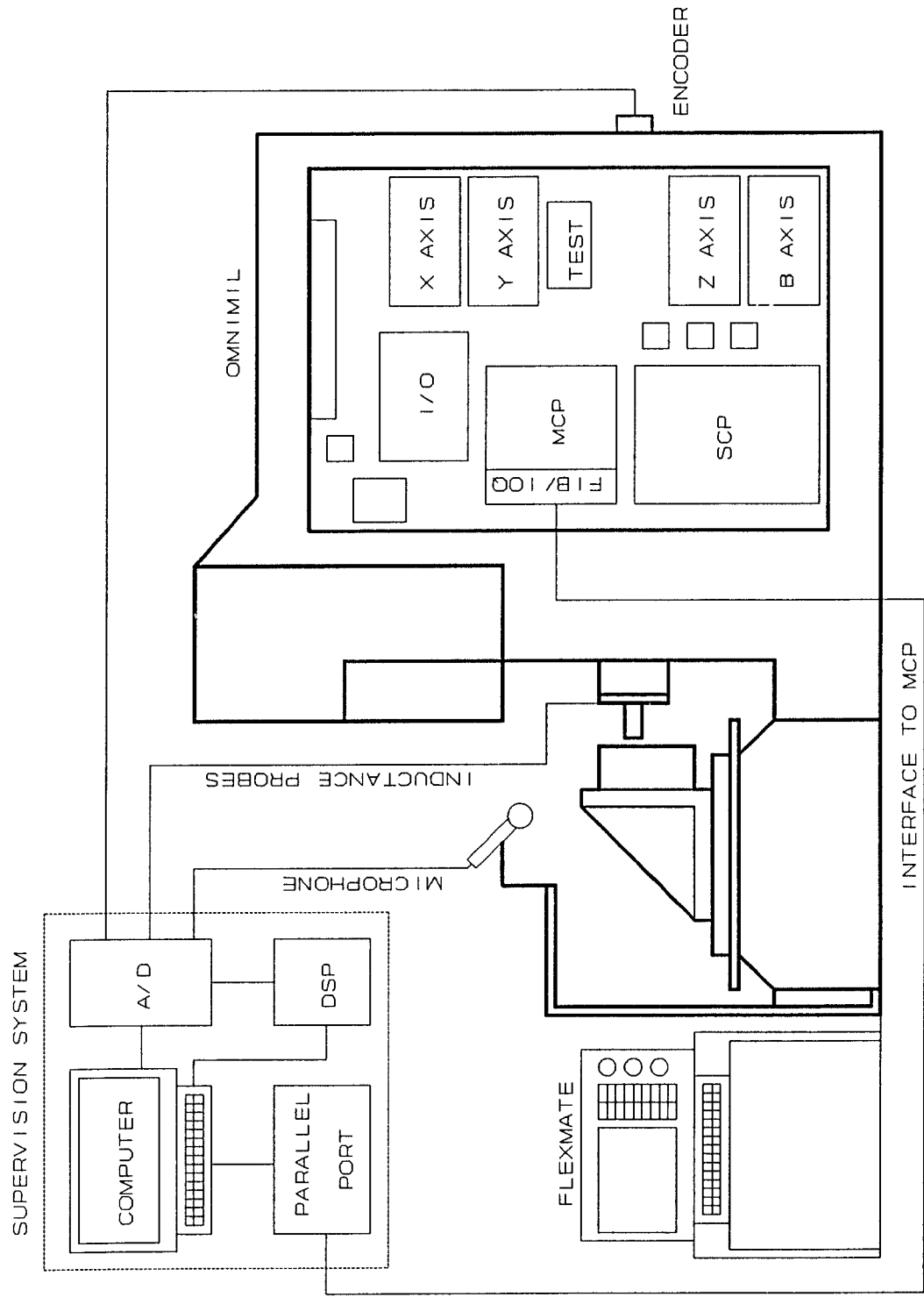


Figure 4.1.1. Diagram of the On-Line Machine Tool Supervision System.

A sensor ring that contains four Mac-Euro inductance probes is used for measuring the X and Y direction vibrations on the Omnimil. Two probes measure the X axis vibration, and two measure the Y axis vibration. The ring is mounted on the spindle quill, and picks up the vibration of the tool holder relative to the quill. It has been shown that this location for a vibration transducer, although it does not directly measure the vibration of the tool relative to the workpiece, gives an accurate indication of the vibrations generated by the metal cutting process (Tarnq, 1988).

The resolution of the inductance probes has been determined to be 17.5 microns per volt, and their bandwidth is 0 to 3000 Hz. The signals from the probes are pre-processed by a Mac-Euro MEB-1210 signal conditioning board, which can also compute the magnitude of the vector sum of the X and Y axis vibrations. The signals are sampled by the Analog-to-Digital data acquisition board in the supervision computer.

The spindle encoder is a BEI Chatsworth model 84C, 240 line, incremental optical encoder with a 1400 cycle resolution and a maximum slewing speed of 5000 RPM. It is attached directly to the spindle shaft in the rear of the Omnimil. The 240 pulse per revolution signal is used as an external clock for the data acquisition. Output of the encoder is also processed through a divide by 240 counter circuit, giving a once per revolution pulse train. This signal is used for external triggering of the data acquisition. The two signals

together can thus be used to synchronize the data acquisition to the rotation of the cutting tools. The once per revolution signal is also used to monitor the spindle speed, as described in the discussion of the torque overload system in Chapter 3.

The microphone used for chatter detection is a Realistic model 33-2011 condenser microphone with a frequency response band width of 20 to 13000 Hz. Machine tool vibrations can be effectively measured from audio signals if the noise of adjacent machines can be excluded (Delio, 1989). The advantage of using a microphone is that it does not need to be physically connected to the machine tool. It is also possible to use the vibration signals from the inductance probes to detect the onset of chatter, but the bandwidth of the probes limits their ability to resolve chatter frequencies above 3000 Hz. Since the microphone signal range is less than 500 mV, a variable gain amplifier is used to boost the signal to the +/- 10 volt range of the data acquisition board.

Data acquisition for the sensor signals is by a Data Translation model DT-2818 A/D-D/A board (Data Translation, 1988). The DT-2818 provides four single ended Analog-to-Digital (A/D) input channels (simultaneously sampled and held), two Digital-to-Analog (D/A) output channels, and two eight bit Digital Input/Output (DIO) channels. It has been configured to have A/D and D/A input ranges of +/- 10 volts. The DT-2818 has 12 bit digital resolution, which means that the voltage resolution is  $20 \text{ volts} / 4096 = .00488 \text{ volts}$ .

The DT-2818 can sample data in three basic modes. In the immediate mode, single data values are read each time the board is activated. In the block mode, a specified number of samples are taken, with the conversion rate governed by the speed of the sampling loop in the host computer. In the Direct Memory Access (DMA) mode, the sampling rate is limited only by the performance of the data acquisition board. The DT-2818 is capable of a maximum theoretical data throughput of 27.5 kHz in the DMA mode. Specific aspects of DMA data acquisition are discussed in the next chapter.

The FFT computations required by the Chatter Recognition and Control system are performed by an Athena DUAL-32 digital signal processing (DSP) board. The DSP board is installed inside the supervision computer, and executes an FFT algorithm that has been coded directly into its processor. The speed of the processor makes it possible to perform real time FFT calculations on windows of data sampled by the DT-2818 board.

As well as sampling signals from the sensors, the supervision system must also monitor parameters from the machine tool itself. The majority of these values are known to the FlexMate controller, and can therefore be read by the supervision subroutines. However, if future development of the system requires sampling of analog signals from the machine tool, such as servo tachometer voltages, then extra data acquisition channels could be provided by adding a second DT-2818 board, or by substituting a data acquisition board that

has a higher channel density (and perhaps a faster DMA data throughput).

Table 4.1 summarizes the sensor and machine parameter inputs that are needed by the supervision system. It can be seen that a relatively small number of sensor inputs are needed to implement the system. Of course, different sensors could be used depending on the requirements of specific control strategies and machine applications.

Table 4.1. Sensor Inputs Required by the Supervision System.

SENSOR	LOCATION	SUPERVISION SCHEME
Encoder	Spindle Shaft	All
Inductance Probe (X and Y Axis)	Spindle Housing	A/C, Broken Tool
Microphone	Near Tool/Workpiece	Chatter Recognition

### The Interface Hardware

As mentioned in Chapter 2, the Fast Input Board (FIB) is supplied with the FlexMate controller as a means for the machine tool user to interface with the Motion Co-Processor (MCP). The FIB is software compatible with the Input/Output Converter Driver Card (IOQ). It is the combination of the FIB and IOQ boards that provides the hardware capability of interrupting and communicating with the FlexMate MCP. The IOQ

and FIB boards communicate with the MCP using the I/O bus inside the MCP card cage (see Figure 2.7).

Figure 4.2 shows a diagram of the interface between the supervisory computer and the MCP in the FlexMate. The physical connections are made through a distribution box that is mounted inside the control enclosure of the Omnihil. The computer, which is a 6 MHz 80286 IBM AT with a numeric co-processor, has a 32 bit Data Translation DT-2817 digital I/O board installed. Ports 0 and 1 are enabled for output and ports 2 and 3 are enabled for input. Each port has eight bits.

Bit 7 of port 1 is connected to bit 0 of the FIB board in the MCP. This is the FIB INTERRUPT line that signals the MCP that the supervisory computer is requesting attention. Since the FIB operates on 24 volt logic, the TTL output of the computer is used to switch a relay circuit that passes 24 volts when activated (Tyler, 1989).

There are four 16 bit ports on the IOQ board. The interface uses only two of them, port D which is enabled for input and port E which is enabled for output. Data transfer is synchronized by the ACKNOWLEDGE bit on the supervisory computer side and the TRANSMIT READY bit on the FlexMate side. Data transfer consists of sending information in single byte (8 bit) packets from port 0 on the computer to port D on the IOQ, and from port E on the IOQ to port 3 on the computer. In the following section, the actual communication protocol will be outlined.

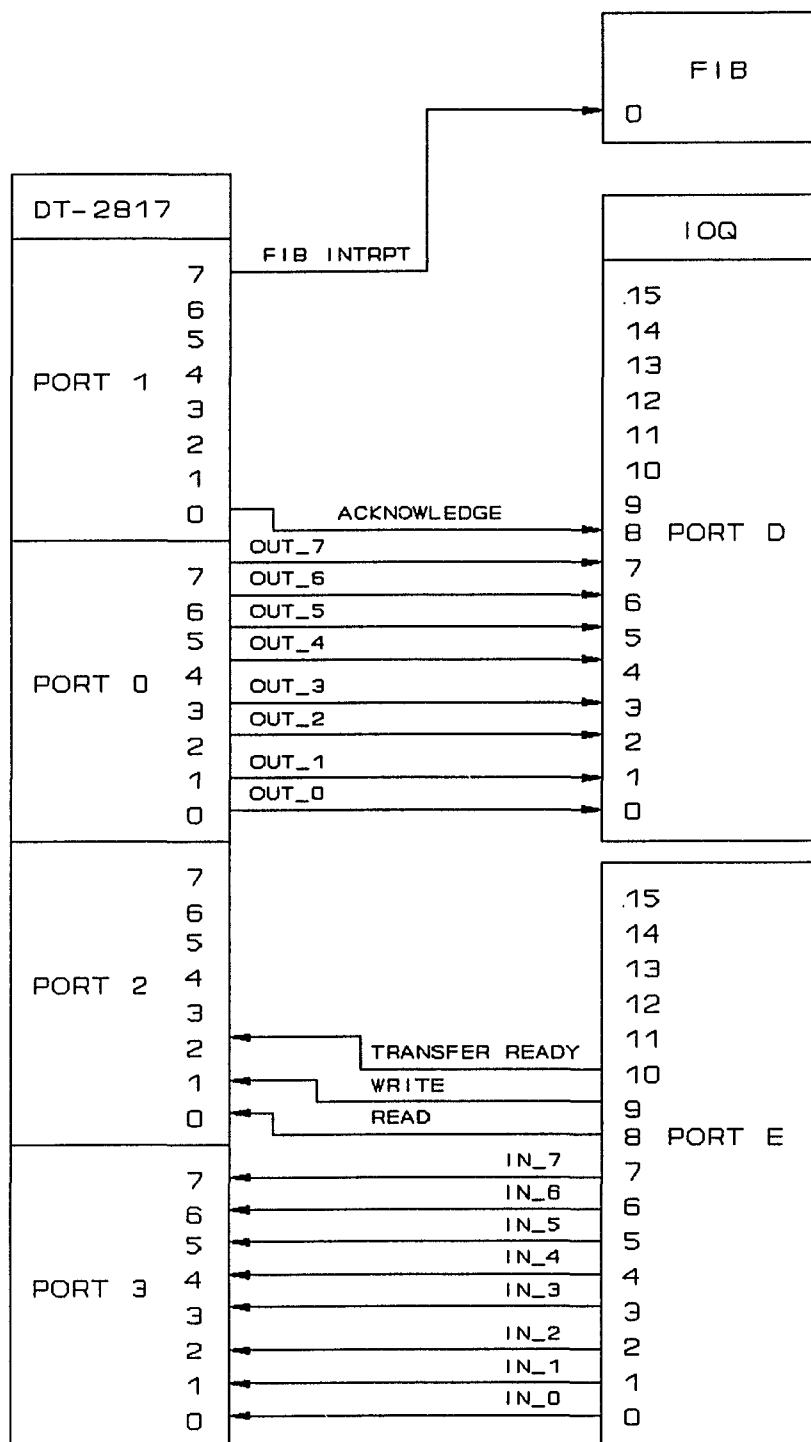


Figure 4.2. Hardware Configuration of the Supervision Interface.

### The Interface Software

The software that handles the interface on the FlexMate side is called `iface.c`, and is installed in the `OEM_MAIN` area of the MCP as described in Chapter 2. The program `pciface.c` is the corresponding interface and supervision software in the supervisory computer. Both programs are listed in Appendix A, and are described in detail in the report by Wells (1991a). Communication is always initiated by the supervisory computer, and the actual data transfer is coordinated by the code running in the MCP. The low-level communication protocol used in the programs was designed by Walters (1991).

With the exception of the `FastStop`, `ClearFastStop` and `SetFeedOvrd` routines, which are handled at FIB interrupt time, a call to one of the supervision subroutines initiates a data transfer between the computer and the MCP. Each supervision subroutine uses the `mcp_call` function:

```
mcp_call (mcp_cmd, out_data_type, out_data, in_data_type,  
          in_data);
```

The first value in the `mcp_call` argument list is a short integer, `mcp_cmd`, which tells the MCP which supervisory routine is being initiated. The second value is a short integer, `out_data_type`, that tells the MCP what type of data is going to be sent (short or long integer, or NULL). The third value, `out_data`, is the data to be sent to the MCP. The fourth value is a short integer, `in_data_type`, that defines



what type of data will be returned from the MCP (short or long integer, or NULL). The fifth value, `in_data`, is the data that is returned from the MCP.

The function `mcp_call` first issues an interrupt to the MCP using the `interrupt_mcp` function. The MCP acknowledges the interrupt by setting the TRANSMIT READY bit high. The command code, `mcp_cmd`, is then written to the MCP, and a switch statement is used to decide what kind of data needs to be written to or read from the MCP.

Since data transfer takes place one byte at a time, it is instructive to describe the sequence by which one byte is transferred from the supervisory computer to the MCP in the `write_to_mcp` and `read_from_mcp` functions:

1. Wait for a TRANSMIT READY signal from the MCP.
2. Write the byte on port 0, or read the byte on port 3.
3. Set the ACKNOWLEDGE bit on port 1.
4. Wait for a TRANSMIT STOP signal from the MCP.
5. Clear the ACKNOWLEDGE bit on port 1.

This sequence is repeated twice for single word (short integer), and four times for a long integer transfer.

Communication on the MCP side is handled in switch statements in the `iface_main` function of `iface.c`. Based on the command word, and the input and output data types, the MCP knows which command it has to execute, and whether there is data to be read from or written to the supervisory computer.

Integration of the Supervision System

The interface has been designed so that additions and modifications to the supervision subroutines can be made relatively easily as the need for new functions is identified. For the present system, however, a discrete set of control options has been provided. Table 4.2 presents a list of the basic machine monitoring and control parameters that are needed by the supervision system. It can be seen that only a few critical control functions are necessary to provide the system the means for managing the machine tool.

Table 4.2. Machine Control Parameters Needed by the Supervision System

CONTROL PARAMETER	INPUT OR OUTPUT	SUPERVISION SCHEME
Fast Stop	Output	All
Feed Rate Percent Override	Input / Output	A/C, Chatter
Spindle Speed	Input / Output	Chatter, Torque

The supervision subroutines created as part of the present research provide an elegant means for the system to communicate with the machine tool controller. Before the supervision system interface was created, control of such values as the spindle speed and the feedrate override was accomplished by testing signals in the controller, and splicing wires to the appropriate circuits. Integration of the

separate schemes into a comprehensive system has meant, in part, replacing the reading and writing of voltages through data acquisition boards with calls to subroutines that command or retrieve the required data. Considering the needs of each scheme, the subroutines listed in Table 4.3 have been created.

Table 4.3. List of the Supervision Subroutines.

SUBROUTINE	FUNCTION
1. Dt2817Init ();	Initialize I/O board
2. FastStop ();	Request fast stop
3. ClearFastStop ();	Clear fast stop
4. ExtFeedhold ();	Request external feed hold
5. SetFeedOvr (int pfp);	Set feedrate override
6. GetFeedrate (long *fr);	Get programmed feedrate
7. GetManFeedOvr (int *mfp);	Get manual feed override
8. SetSpindleSpeed (int ns);	Set the CNC spindle speed
9. GetSpindleSpeed (int *rpm);	Get the CNC spindle speed

The Dt2817Init routine must be called at the start of a supervision program to initialize the DT-2817 digital I/O board. The FastStop and ClearFastStop routines are discussed below. The ExtFeedhold routine toggles an external feedhold on the MCP. This kind of feedhold can be cleared by the machine operator, and is useful for suspending machining in a non-emergency situation.

The SetFeedOvr routine allows the programmed feedrate override to be changed up to 200 percent from the supervisory computer. The argument is a short integer representing the feed override as a percentage. This subroutine does not initiate a data transfer to the MCP, but sets the programmed

override at the time of the FIB interrupt. The feedrate override is then maintained as part of the stop.c program that is running in OEM\_SYNC. Also, completion of the command is not acknowledged, but rather the FIB interrupt bit is held high in the supervisory computer for approximately 0.005 seconds and then cleared. The subroutine was implemented in this way because the Adaptive Control scheme required a fast real time response to feedrate override commands for stability of the control loop. This is further discussed in Chapter 6.

The GetFeedrate routine returns the current feedrate from the MCP as a long integer in the units of inches per minute. The GetManFeedOvrdr routine returns the manual feed override setting from the operator console as a short integer with the units of percent. In order to know the net commanded feedrate, both the manual and programmed override percentages must be applied to the current feedrate returned by GetFeedrate.

The SetSpindleSpeed routine allows the spindle speed to be changed from the supervisory computer. The argument is a short integer representing the new speed in RPM. The GetSpindleSpeed routine returns the current net commanded spindle speed, including the manual percent override, as a short integer with units of RPM.

The FastStop routine, described in Chapter 3, is used by all the supervision schemes, but is most critical in the Adaptive Control scheme when the tool impacts the workpiece and in the Chatter Regulation scheme when the machine tool is

a severe state of chatter. The fast stopping routine can stop both the X and Y axes in as short a time as 0.035 seconds at a feedrate of 50 in/min. It will be shown in Chapter 6 that the stopping time increases with the feedrate.

At the time of the fast stop, an internal feedhold command is issued to the MCP. This stops interpolation. Then the output from the CNC to the servos is severed. These steps take place at the time of the FIB interrupt. A one percent feedrate override is commanded, and the fast stop is held active, in `stop.c` (`OEM_SYNC`) until the `ClearFastStop` subroutine is called. At clear time, the axes are moved slowly to eliminate the remaining CNC positional error by writing voltages directly to the servos. This means that some residual motion will always be completed before the internal feedhold is cleared. Releasing the internal feedhold enables the commanded axis motion to resume. The completion of both the `FastStop` and `ClearFastStop` subroutines is acknowledged by the MCP in a function that sets the TRANSMIT READY bit, waits for the ACKNOWLEDGE bit and then clears the TRANSMIT READY bit.

Figure 4.3 shows a schematic of the overall supervision system. The flow of control data between the supervisory computer and the machine tool is indicated by the arrows connecting the elements of the system. It can be seen that the supervision subroutines comprise the heart of the system.

Table 4.4 lists which subroutines are needed by each monitoring and control scheme. It can be seen that the

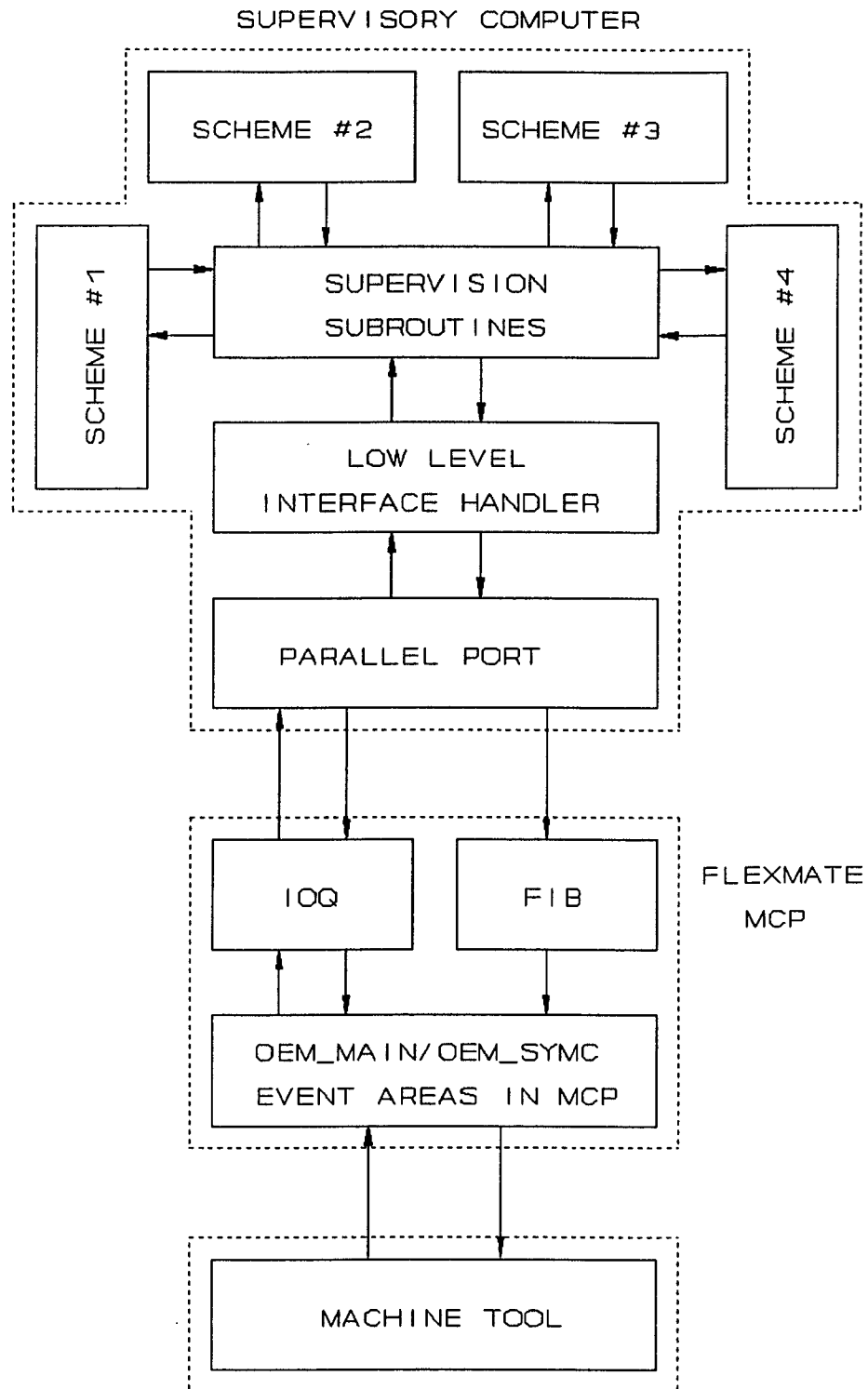


Figure 4.3. Schematic of the Supervision System and the FlexMate Motion Co-Processor.

Adaptive Control and Chatter Regulation schemes make the most extensive use of the interface. The Broken Tool Detection scheme and the Spindle Torque Overload scheme both need only the fast stopping routine.

Table 4.4. List of the Supervision Subroutines Used by Each Scheme of the Supervision System.

SUPERVISION SCHEME	SUBROUTINES USED
Adaptive Control	1, 2, 3, 5
Chatter Regulation	1, 2, 3, 5, 8
Broken Tool	1, 2, 3
Torque Overload	1, 2, 3

For the present research, each monitoring and control scheme in the supervision system will be exercised one at a time. The use of parallel or distributed processing strategies to run the schemes concurrently is presently being studied in the Machine Tool Laboratory.

Each scheme essentially consists of a computer program, and the necessary sensor and data acquisition hardware. The supervision subroutines are compiled into a library, and the routines are available to all of the schemes. Integration of the schemes into a comprehensive system has consisted mostly of three steps. First, the supervision programs were either modified or rewritten so that they could be linked to the library of supervision subroutines. It should be noted here that the supervision subroutines were written in Quick-C, and

that programs written in either Quick-C or Quick-Basic require no modification to make use of the routines. Second, the input/output routines that provided access to the machine tool, which were usually implemented by reading and writing voltages using a data acquisition board, were replaced by calls to the appropriate supervision subroutine. Third, the schemes were recalibrated to make use of the new sensors, consisting essentially of the inductance probes and the spindle encoder, used for the supervision system.

In the next chapter, an off-line system for the characterization and evaluation of machine tools will be presented. This system, created as part of the present research, was used extensively in the development and testing of the on-line system. The implementation of the on-line supervision schemes is described in Chapter 6.



## CHAPTER 5 THE OFF-LINE MACHINE EVALUATION SYSTEM

This chapter describes an off-line machine evaluation system. It consists of a portable computer, a data acquisition board, sensors, signal conditioning hardware and the system software PCDATA. The purpose of the system is to allow identification of the dynamics of production machinery, as well as the monitoring of transducer signals. The off-line system was used to develop, modify, and test the monitoring and control schemes used in the on-line machine supervision system.

### Hardware Requirements

PCDATA is a program, written in Quick-Basic, that consists of several modules. A listing of the program is presented in Appendix C, and the program is discussed in detail in the report by Wells (1991b). The Data Acquisition Module allows sampled data to be plotted on the screen and saved to disk. A Fast Fourier Transform (FFT) can be performed on the data and the spectrum can be plotted on the screen. The System Test Module can be used to both read and write single voltages, and is handy for monitoring transducer signals.

There is also a Data Processing Module that can be used to plot and FFT data that has been previously stored on disk. This allows data to be saved and evaluated at a later time. The Data Acquisition Module, Transfer Function Module and Chatter Analysis Module will be described in the following sections.

Figure 5.1 shows a diagram of the system hardware. Any kind of voltage signal can be sampled by the Data Acquisition Module of PCDATA. For the Transfer Function Module, it is expected that an impact hammer will be used for the input, and either an accelerometer or a displacement transducer will be used for the response. Input to the Chatter Analysis Module of PCDATA is expected to be a microphone, which must be amplified to an appropriate voltage range. Two of the data acquisition channels have phase matched 12 kHz passive low pass filters to eliminate signal aliasing. The system was installed on a portable 80386 33 MHz machine with an 80387 math co-processor.

Data acquisition is accomplished using a Data Translation DT-2818 A/D-D/A board (Data Translation, 1988). It provides four single ended A/D input channels, which are simultaneously sampled and held. This means that the inputs on all the channels are sampled at the same instant, and there is no phase lag between input and response measurements.

The DT-2818 also has two D/A output channels and two eight bit DIO channels. It has a maximum theoretical A/D data throughput in the Direct Memory Access mode of 27.5 kHz. In

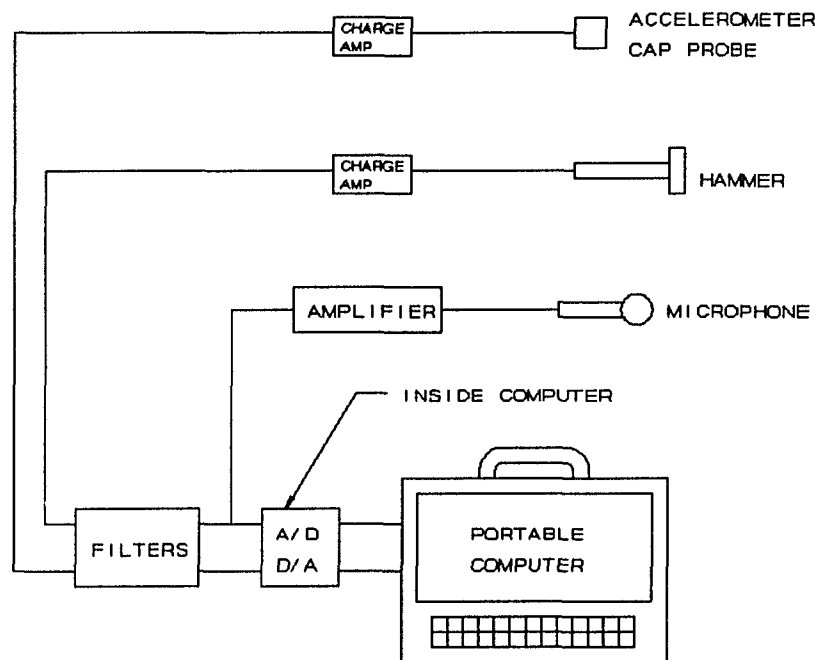


Figure 5.1. Hardware Diagram of the Off-Line Machine Evaluation System.

practice, a maximum sampling rate of 25 kHz can be achieved. The board has been jumpered to have an A/D input range of +/- 10 volts and a D/A output range of +/- 10 volts. The DT-2818 has 12 bit digital resolution, which gives it a resolution of  $20 \text{ volts} / 4096 = .00488 \text{ volts}$ .

The machine evaluation software uses the Direct Memory Access (DMA) mode of data acquisition, which is activated by programming the Intel 8237 DMA controller chip that is on the mother board of all IBM PC/AT and compatible computers. The advantage of the DMA mode is that the maximum rate of data transfer can be achieved, since the DMA controller sends the data as bytes directly to memory (without the intervention of

the particular computer's microprocessor and therefore without the processing speed of the microprocessor being a factor).

Data can be written to or read from memory under DMA control using two different modes. The "single byte" mode transfers only the specified number of data conversions to memory. The "autoinitialize" mode will continue transferring data, overlapping the earliest data in memory in a circular buffer fashion, until a stop command is issued. The number of data conversions is written to the byte count register of the DMA chip, and defines the size of the DMA data buffer within the memory page. For example, 25000 data conversions would allocate 50000 bytes of memory as the DMA buffer, since each discrete integer datum generated by the DT-2818 (0 to 4096) occupies two bytes.

Using the DMA chip also requires programming corresponding features on the DT-2818 board. The non-continuous block read mode performs a specified number of data conversions. This mode is used in the Data Acquisition Module. The continuous block read mode performs data conversions continually until a stop command is issued to the DT-2818. This mode is used in the Transfer Function Module because of the software pre-triggering it allows.

An issue in DMA data transfer is the location in the computer's memory where the data will be stored. Memory is structured into 64K (65536) byte blocks referred to as DMA pages, with page 0 being the lowest. The DMA page must be

specified in software when programming the DMA controller chip, and must not conflict with DOS and other resident system programs. Also, the amount of data stored, which may be less than 64K bytes, must not be greater than 64K bytes and must not cross a DMA page boundary.

### Data Acquisition

The Data Acquisition Module of PCDATA allows up to four channels to be sampled for data acquisition. The first channel sampled will always be channel 0. The user then enters the calibration numbers for the transducers that are connected to the selected channels. These scaling factors allow the data to be expressed in engineering units. He then selects the number of samples per channel. The maximum total number of samples has been defined to be 25000. If four channels were being read, the maximum number of samples per channel would be 6250.

The fastest attainable DMA sampling rate for one channel using the DT-2818 board is 25000 Hz. The simultaneous sampling of the DT-2818 board requires that the total sampling rate be divided by the number of channels being used, so the maximum possible sampling rate on four channels would be 6250 Hz. By changing the samples per channel, and the sampling frequency, the user can control the total observation time.

After the data has been acquired, it must be extracted from the memory page where the DMA chip has placed it. A PEEK

loop is used to extract the low and high bytes of the data from memory. The integer data values are converted into scaled real values, and then assigned to their respective data arrays using modulo division to increment the channel numbers.

When it has been recovered, the time domain data for each channel can be plotted, or an FFT of up to 8192 points can be calculated from the data (using a C language linked subroutine, translated from Press et al., 1986, which employs the Cooley-Tukey algorithm). The size of the FFT can be defined through the program's Setup Module. If fewer data points have been taken for each channel than the selected FFT size, then an FFT cannot be computed. If more data points have been taken, then the user can select a time window from the full data record for the transform. The time data and spectrum for each channel are then plotted. The features available for plotting are described in a help line that appears at the bottom of the graphics screen. The program also supports a screen dump of the plots in either a dot matrix or laser printer format.

A principal value of the PCDATA program is that the time domain data can be saved to disk for later processing. Using a binary file format, the entire time record for each channel is saved. The binary file approach is used to minimize both disk access time and the size of the data files. Binary data retrieved by the Data Processing Module can in turn be saved in ASCII format for processing by other software.

An important consideration in data acquisition is signal aliasing. This phenomenon can occur when the data contains a frequency component that is either above the sampling frequency of the data acquisition system, or is between the sampling frequency and the maximum frequency that can be seen in the FFT spectrum. Dynamic analyzers, such as the GenRad 2515 computer test system (GenRad, 1985), include anti-aliasing low pass filters that remove alias frequencies from the sampled data.

As an example of signal aliasing, Figure 5.2 shows a plot of a 100 Hz signal, and its spectrum, sampled by the Data Acquisition Module of PCDATA at a rate of 5000 Hz. Another frequency of 8000 Hz, 3000 Hz above the sampling frequency, was added to the signal using a second function generator. It can be seen that the higher frequency has aliased into the spectrum at 2000 Hz.

Since it is not usual to know beforehand the frequency content of a time domain signal, the alias frequency could be mistaken as a real characteristic of the system being measured. Figure 5.3 shows a plot of the same signal taken through a Wavetek model 432 low pass filter with its cutoff frequency set at 5000 Hz. It can be seen that the spectral line at 2000 Hz has now been almost fully attenuated.

Figures 5.2 and 5.3, which demonstrate the data taking capability of PCDATA, show that it is essential to properly condition signals before sampling them. The off-line machine

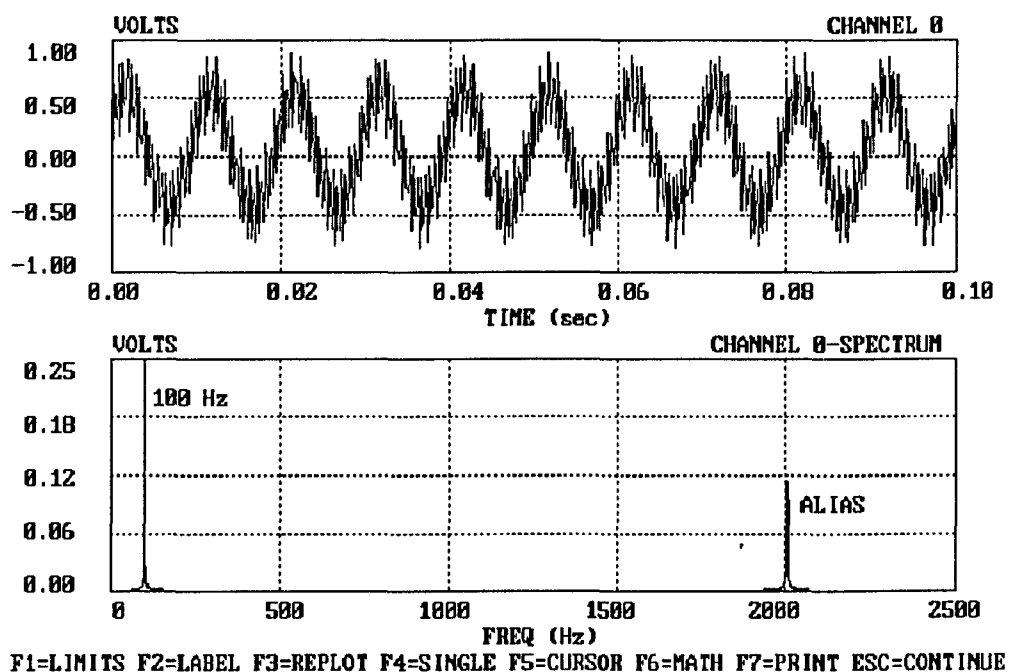


Figure 5.2. Time Data and Spectrum Plots of an Unfiltered 100 Hz Signal with Aliasing.

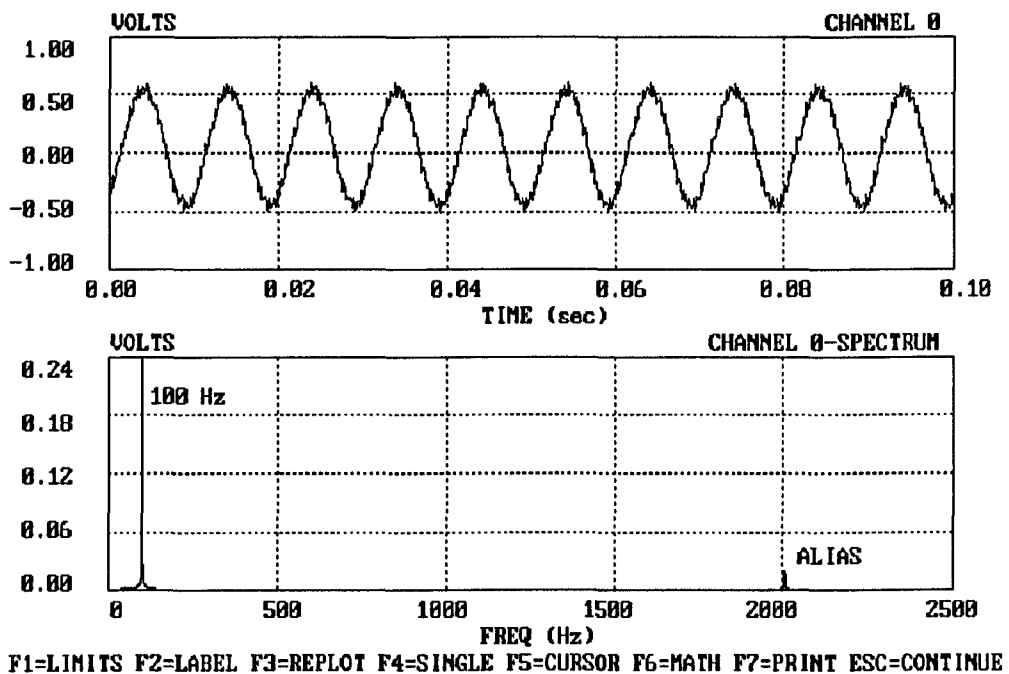


Figure 5.3. Time Data and Spectrum Plots of a Filtered 100 Hz signal with Aliasing.



evaluation system includes two phase matched passive low pass filters with cutoff frequencies of 12 kHz. This cutoff frequency was selected because high frequency signals are rejected, and because frequencies up to 12.5 kHz can be seen in the spectrum if one channel is sampled at 25 kHz using the Data Acquisition Module.

### Experimental Modal Analysis

The Transfer Function Module of PCDATA offers capabilities similar to some of those provided by the GenRad 2515 computer test system (GenRad, 1985) for measuring Transfer Functions. The user begins by selecting a sampling frequency for the Transfer Function (TF) channels. He can select up to 12000 Hz, or choose a lower sampling rate. The latter choice may be sometimes useful in order to obtain a finer frequency resolution in the spectrum, since

$$df = sfreq / n \quad (5.1)$$

where  $df$  is the frequency resolution,  $sfreq$  is the sampling frequency and  $n$  is the number of samples taken.

The excitation signal for the TF, which will always be an impact hammer, must be on channel 0 of the DT-2818 board. This is because only channel 0 is monitored for the trigger. The

response, which will always be channel 1, is usually measured either as displacement or acceleration.

The trigger threshold is a value, in volts, above the DC offset of the hammer charge amplifier. It has been found that 0.2 volts works well for most situations. For lighter hits, the gain on the hammer charge amplifier can be increased. In order to determine a threshold for the hammer, the Data Acquisition Module can be used to plot several test hammer hits so that the hammer voltage can be evaluated. The threshold value is referred to the DC offset of the hammer charge amplifier by summing it with the voltage being output on channel 0. This value is in turn used to define a discrete limit value, whose range is from 0 to 4096, that is computed each time the program waits for a hammer hit.

In order to produce a reliable TF, it is necessary to average some number of hits. It has been found that 3 to 5 averages are sufficient for accurate results. After the data acquisition process has been begun, the message "Waiting for trigger # 1 (Esc to Exit) ..." appears in the center of the screen, and the command to read A/D with continuous DMA is issued to the DT-2818. In the background, the data transfer is begun. The DMA buffer size for the Transfer Function Module has been set to 32000 values (16000 values for each channel) which makes the buffer 64000 bytes long in memory. The data from channel 0 and channel 1 are streamed continually into a circular memory buffer defined for the DMA chip.

The PEEK statement is used to extract values from the memory locations corresponding to data from channel 0. If the value returned exceeds the threshold limit value, then a hammer hit is detected. This software triggering makes the program dependent on the speed of the computer used for the off-line system, but provides the pre-triggering necessary to capture the entire hammer impulse. Once the trigger has been found, the program polls the DMA address register and stops the DMA process when the number of data points required for the FFT have been taken. The hit is pre-triggered by taking a point three data values prior to the sensed trigger as the actual start of the data window.

After the trigger has been detected, the message "Processing (ESC to Hold) ..." appears on the screen. The data that has been stored in memory by the DMA transfer is now extracted, scaled, and assigned to the appropriate channel data arrays. A square window, 120 data values long, is applied to the hammer signal in order to remove noise from the hammer spectrum. It was found that 120 data values are sufficient to detect double hammer hits. If the data wraps around the DMA page boundary then two loops are used to extract the data. The comments in the code listing of PCDATA in Appendix C explain the details of the data extraction algorithms.

Next, the data from both channels are averaged, and the average is subtracted, datum by datum, while an exponential window is applied to the data. Subtracting the average gives

the effect of removing the DC offset from the data, and the exponential window greatly improves the quality of the signals by attenuating noise that continues in the transducer signals after the transient from the impact has died out. These two steps are represented by the following expression

$$\text{DATA}(J) = (\text{DATA}(J) - \text{AVG}) * \text{EXP}(-J * \text{DT} / \text{TAU}) \quad (5.2)$$

where DATA() is the data array, J is a loop counter, AVG is the average value of the data, DT is the time step of the data sampling, and TAU is the time constant for the exponential window. A value for TAU of one tenth of the total observation time was found sufficient to attenuate noise while not corrupting the impact signal.

If a voltage overload is detected on either channel then the warning "OVERLOAD! Continue or Hold? [C/\*H]" is printed on the screen. Pressing RETURN or H will cancel the hit, and the program will return to waiting for a trigger. Pressing C allows the user to take the hit in spite of the overload.

The real and imaginary values of the average TF are computed using the Cross and Auto Spectrum (GenRad, 1985). The Auto Spectrum calculates the average of the squared magnitude of the hammer spectrum, representing the mean power of the impact at each frequency. The Cross Spectrum calculates the averaged complex product of the hammer spectrum and the response spectrum. It indicates which frequencies match

between the two spectra. The Transfer Function is then calculated as the ratio of the cross spectrum of the impact and response to the input power spectrum of the hammer hit. Writing the above relationships mathematically gives the following expressions

$$\text{Auto Spectrum} \quad G_{aa} = S_a \cdot S_a^* \quad (5.3)$$

$$\text{Cross Spectrum} \quad G_{ab} = S_b \cdot S_a^* \quad (5.4)$$

$$\text{Transfer Function} \quad H_{ab} = G_{ab} / G_{aa} \quad (5.5)$$

where  $S = \text{REAL} + j * \text{IMAG}$  is an operator representing the real and imaginary parts of a signal, (a) represents the hammer, (b) represents the response and (\*) indicates the complex conjugate. Expanding the above expressions for the real and imaginary parts of the Transfer Function gives

$$\text{Re}[H_{ab}] = (\text{Re}[a] * \text{Re}[b] + \text{Im}[a] * \text{Im}[b]) / \text{Mag}[a]^2 \quad (5.6)$$

$$\text{Im}[H_{ab}] = (\text{Re}[a] * \text{Im}[b] - \text{Im}[a] * \text{Re}[b]) / \text{Mag}[a]^2 \quad (5.7)$$

The user is presented with plots that show the spectrum of the hammer hit and the current average imaginary Transfer Function. The prompt "Continue, Hold, Stop or Average? [\*C/H/S/A]" is printed at the top of the graphics screen. Based on what he sees in the spectrum, the user can continue with the next hit, reject the hit and try again (Hold), stop taking the Transfer Function and return to the module input

screen, or force the average TF to be computed based on the number of hits completed. If hold is selected, the present TF is subtracted from the average TF before the program branches back to wait for the trigger again. When all the hits have been completed, or Average has been selected, the real and imaginary plots of the average TF are displayed.

If an accelerometer has been used for the response transducer, the TF can be converted from acceleration to displacement by dividing by  $-\omega^2$  across the frequency range. This option is supported under the F6=MATH option on the plot screen, as well as the ability to change the scale factors for both the Y and X axis data values.

To verify that the Transfer Function Module of PCDATA gives accurate results, a number of measurements were carried out using the GenRad computer test system as a benchmark. Figure 5.4 shows a GenRad plot of a Transfer Function taken on a 4 fluted 0.75 inch diameter HSS endmill on the Omnihil using a medium hammer and a small accelerometer. Figure 5.5 shows a plot of the same measurement taken by the Transfer Function Module of PCDATA. It can be seen that the Transfer Functions agree well to about 3600 Hz, beyond which frequency the energy from the hammer was insufficient to excite the system (the sampling rate of the DT-2818 board is not fast enough to measure hits with a smaller hammer). Similar correlation was found between GenRad and PCDATA in the many test measurements that were performed during the development of the program.

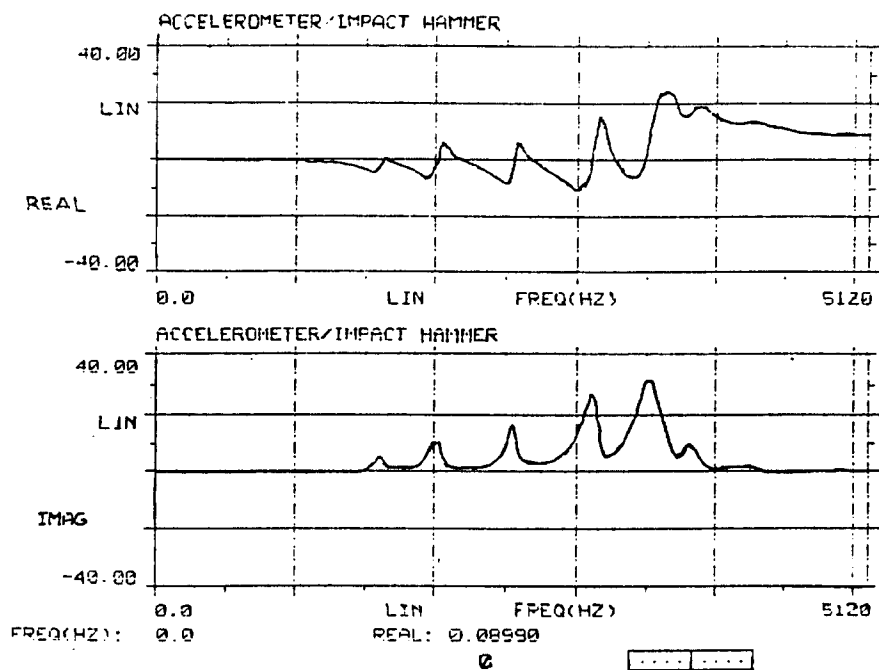


Figure 5.4. Plot of a Transfer Function Measured using GenRad.

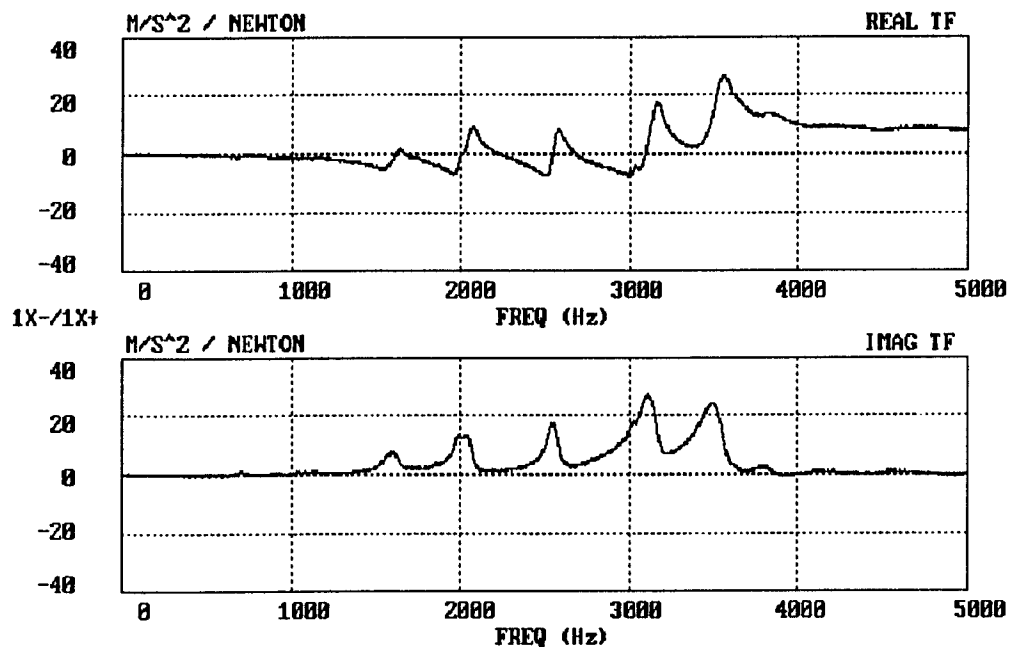


Figure 5.5. Plot of a Transfer Function Measured using the PCDATA Transfer Function Module.

It can be seen that many aspects of the milling process can be measured with the features available in the Data Acquisition Module and the Transfer Function Module of PCDATA. The output of various transducers can be sampled, and Transfer Functions can be taken to identify the structural dynamics of the machine and its various tools. In the next section a specialized module of PCDATA is presented that can be used by the machine operator to analyze chatter.

### Chatter Detection and Analysis

Using the Transfer Function Module of PCDATA, the TF between the tool and the workpiece of a machine tool can be measured. The TF will show the modes in which the tool/spindle system will vibrate. Tlusty (1985) described how the chatter frequency will be near the frequency of one of these modes. The mechanism which causes chatter was described briefly in Chapter 1, and is depicted in Figure 1.3.

Smith (1987) demonstrated that adjusting the spindle speed such that the fundamental tooth frequency is made equal to an integer division of the chatter frequency disrupts the regeneration of waviness that is responsible for chatter. This technique directs the spindle speed into one of the pockets of stability shown in the lobing diagram of Figure 1.5.b.

The Chatter Analysis Module of PCDATA identifies the frequency of machine tool chatter. This information can be



used to calculate a stable spindle speed at which to continue the cutting process. The operator enters the number of teeth on the cutter, the actual spindle speed used for the data sample, and a threshold for chatter detection. He then takes a sound data sample of the unstable cut using the microphone. The module computes and plots the frequency spectrum of the data, and identifies the tooth frequency based on the spindle speed and the number of teeth. The peaks belonging to the tooth frequency and the chatter frequency are identified on the plot.

Figure 5.6 shows the time data and spectrum of an unstable machining pass made on the Omnimil cutting aluminum. The plot is from the Data Acquisition Module of PCDATA. The tool is the same 4 fluted 0.75 inch diameter HSS endmill for which the Transfer Function was presented in Figures 5.4 and 5.5. The feed for the cut was 70 inches per minute, the actual spindle speed, as measured by a photo tachometer, was 2948 RPM and the axial depth of cut was 0.400 inches.

Figure 5.7 shows a plot of the spectrum of the same unstable cut as presented by the Chatter Analysis Module. The chatter peak that is identified at 3623 Hz corresponds to the 3600 Hz mode seen in the TF in Figure 5.5. The program has calculated the tooth frequency,  $F_t$ , and has identified the chatter frequency,  $F_c$ , by locating in the spectrum the largest spectral line that is not a multiple or divisor of the tooth

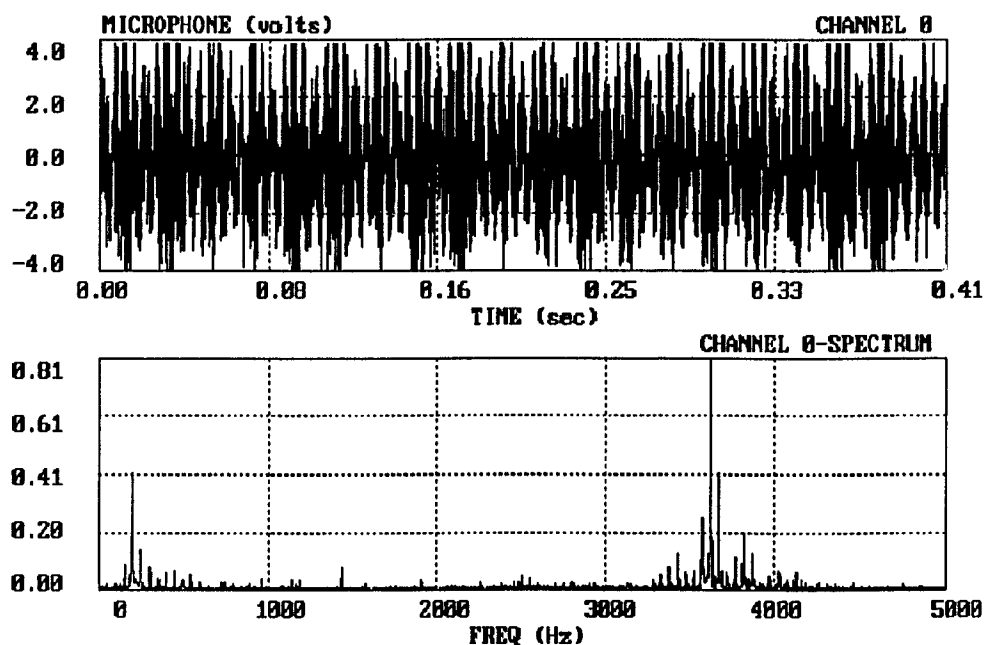


Figure 5.6. Time Domain Data and Spectrum of an Unstable Machining Cut.

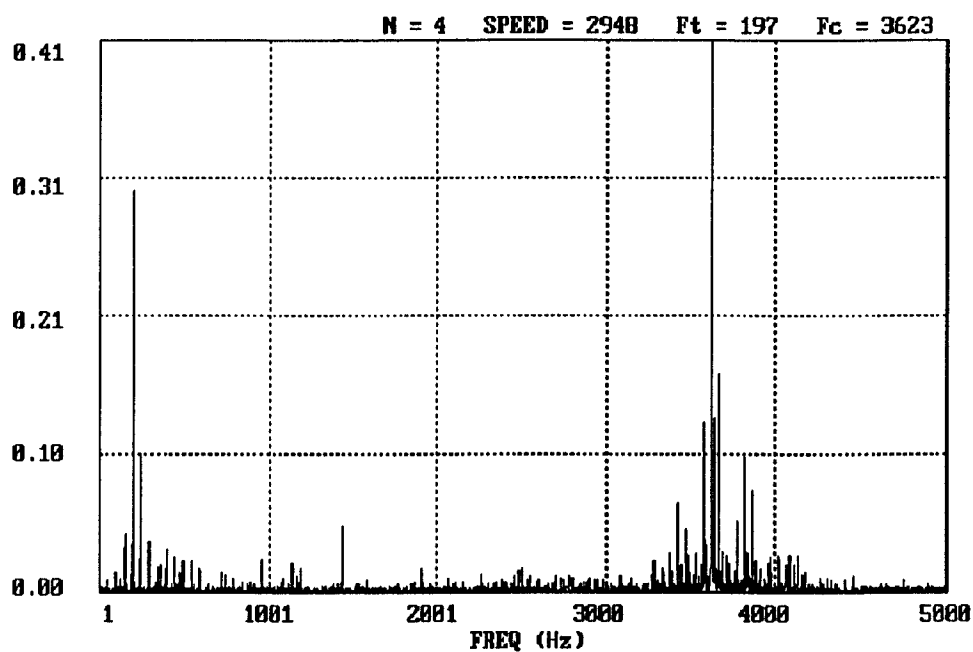


Figure 5.7. Output of the Chatter Analysis Module of PCDATA Showing the Tooth Frequency and Chatter Frequency of an Unstable Cut.

frequency. With this information, a new cutting speed can be calculated from the following expression (Smith, 1987; Delio, 1989)

$$S = (F_c * 60) / (m * (N + 1)) \quad (5.8)$$

where  $S$  is the new spindle speed in RPM,  $F_c$  is the chatter frequency in Hz,  $m$  is the number of teeth on the tool and  $N$  is an integer number representing the number of waves on the machined surface between consecutive cutter teeth.

Since regions of stability repeat in integer multiples,  $N$  can be selected to adjust the new speed to a range appropriate for a given machine tool. However, regions of stability become less well separated at lower spindle speeds (see Figure 1.5.b). This means that the ability to direct the speed into a pocket of stability is governed to some degree by the spindle speed range of a particular machine tool. Also, because chatter can occur in more than one mode, it is possible that more than one iteration of the test may be needed before an absolutely stable speed can be selected.

For the Chatter Analysis Module to operate effectively, it is important that the spindle speed of the machine tool be accurately known. A small error in the spindle speed will be multiplied as the chatter detection algorithm searches across the frequency range for peaks that are not multiples of the fundamental frequency. An 8192 point FFT is used in the

module, with a sampling rate of 16384 Hz. This gives a frequency resolution of 2 Hz. As each peak in the spectrum is evaluated, a range of +/- 3 spectral lines (12 Hz) is tested. This helps keep an error in the spindle speed from causing a false identification of chatter, but also limits the module's ability to resolve chatter when the harmonics of the spindle runout are separated by less than 12 Hz.

In the next chapter, results of a variety of machining tests on the Omnimil are presented that evaluate the supervision subroutines and verify the performance of the on-line supervision system. The off-line system was used for all the data acquisition and Transfer Function measurements required during the tests.

## CHAPTER 6 EXPERIMENTAL VERIFICATION OF THE ON-LINE SYSTEM

In this chapter the performance of the supervision subroutines will be quantified, and the monitoring and control schemes that comprise the on-line supervision system will be demonstrated. The program PCDATA, described in the previous chapter, was used for data acquisition and the measurement of Transfer Functions.

### The Fast Stopping Routine

Since each of the monitoring and control schemes makes use of the fast stopping routine, this function was evaluated first. There are three principal constraints on the operation of the fast stop. First, the time required for the axes to stop moving is directly related to the feedrate. This is to be expected due to the inertia of the drives. Second, the axes must be moved slowly to eliminate the following error that remains after the stop. This is presently accomplished at the time when the fast stop is cleared. Third, the size of the following error remaining after the fast stop is also related to the axis feedrate. Depending on when the fast stop occurs in relation to the last increment from the interpolator, the

value of the CNC following error can jump by over 200 percent, and the MCP will shut down the Omnimil if the following error exceeds 0.500 inches. This was observed when the fast stop was applied at feedrates faster than 250 in/min.

Figure 6.1 shows a plot of the Y axis tachometer voltage during the execution of a conventional feedhold commanded through the ExtFeedhold subroutine. The Data Acquisition Module of PCDATA was triggered when the FIB interrupt line went high. The axis was moving in the negative direction at 50 in/min. It takes about 0.100 seconds for the servo to begin deceleration, and about 0.224 seconds for the axis motion to stop completely. Figure 6.2 shows the results of a fast stop commanded to the Y axis drive at the same feedrate. Again, the data acquisition was triggered at the time of the FIB interrupt. It can be seen that commanding zero velocity to the servo brings the drive to a stop in approximately 0.035 seconds.

Table 6.1 presents a list of the times required to complete a fast stop, and a feedhold, at several different feedrates. As expected, the fast stop out-performs the feedhold dramatically, but the results also show that the fast stopping time does get longer as the feedrate increases. The values of the CNC following error remaining at the time of the fast stop are also listed in the table. As mentioned above, these values vary depending on when the fast stop occurred in relation to the last increment output by the interpolator.

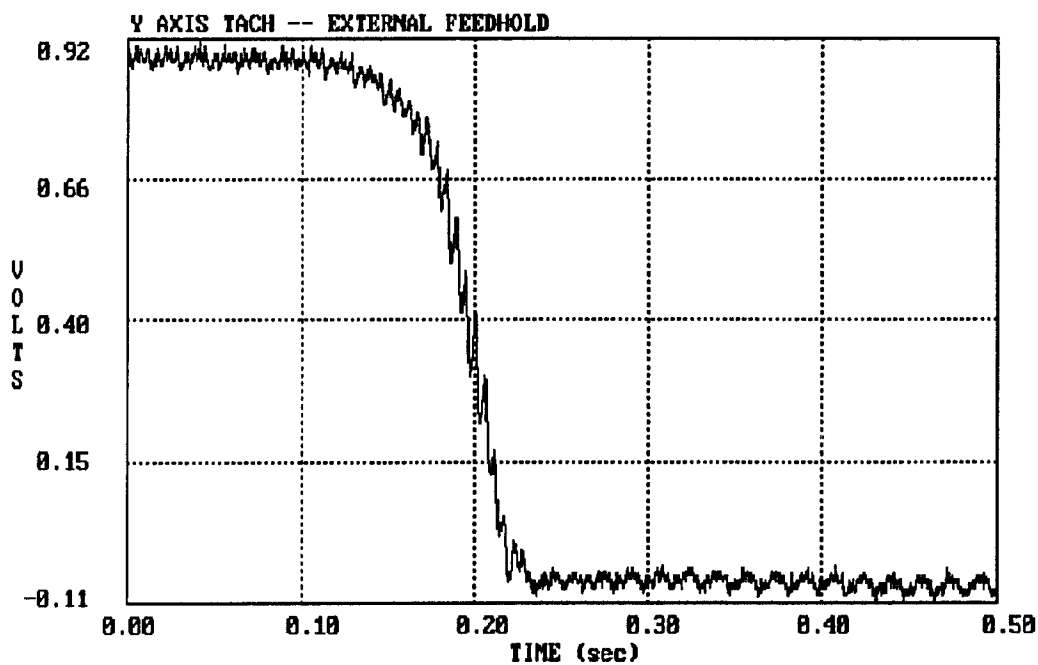


Figure 6.1. Output of the Y Axis Tach During a Feedhold.

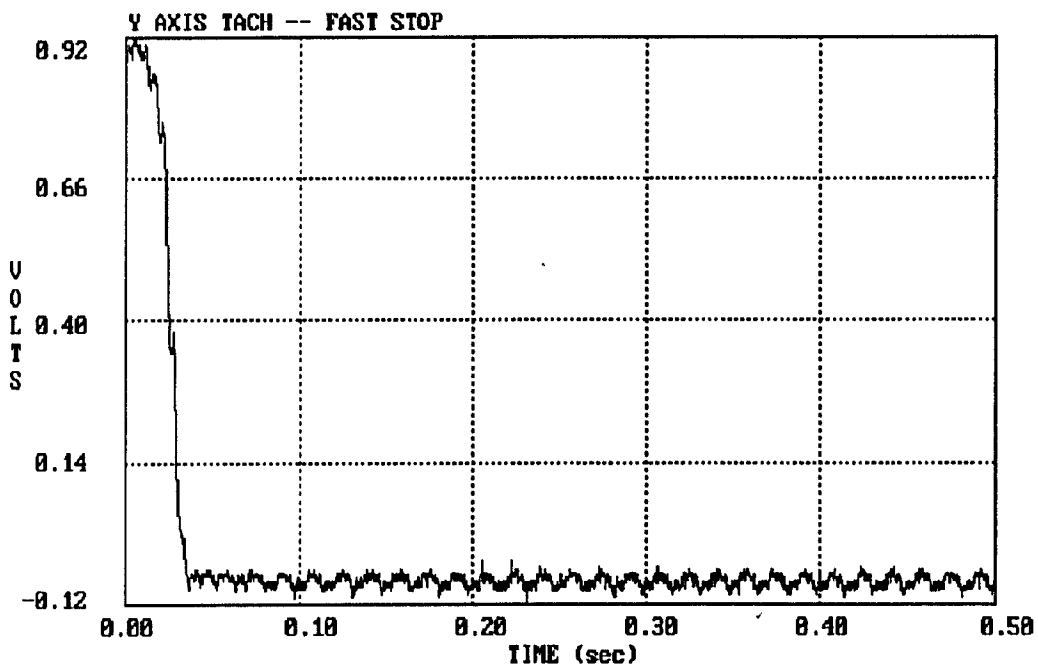


Figure 6.2. Output of the Y Axis Tach During a Fast Stop.

Table 6.1. Response Times for the Fast Stopping Routine and a Conventional Feedhold.

FEEDRATE (in/min)	FEED HOLD TIME (sec)	FAST STOP TIME (sec)	FOLLOWING ERROR (in)
25	0.137	0.032	0.018
50	0.224	0.035	0.036
75	0.308	0.050	0.054
100	0.384	0.052	0.072
125	0.465	0.063	0.089
150	0.547	0.063	0.107
175	0.639	0.072	0.125
200	0.733	0.077	0.142
225	0.794	0.077	0.163
250	0.886	0.078	0.184

Figure 6.3 shows a plot of the following error voltage,  $E_p$ , during a sequence when a fast stop was commanded to the Y axis drive, and then cleared about one second later. The feedrate was 50 in/min. Point A on the plot shows when the fast stop was executed. The  $E_p$  voltage was set to zero at the instant the servo was disconnected from the CNC. The fast stop was cleared at point B. The small voltage between B and C moves the servo to eliminate the CNC following error. At point C the axis is within the +/- 0.001 inch in-position zone established in stop.c, and the servo loop is reconnected. The positional loop brings the servo exactly into position, and the commanded motion is fully resumed at point D.



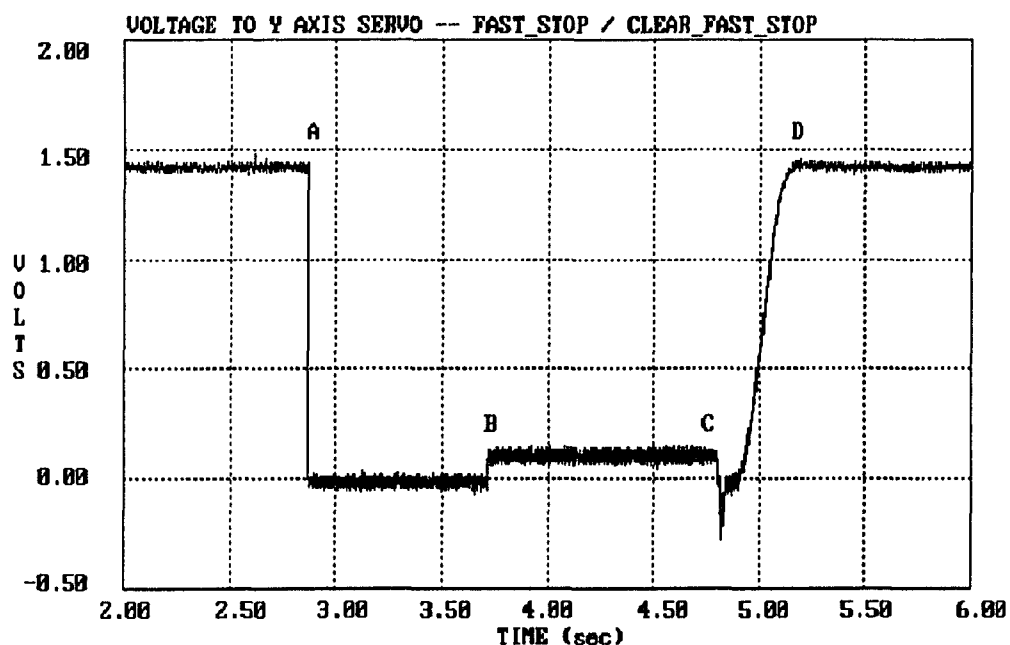


Figure 6.3. Voltage Output to the Y Axis Servo During a FastStop / ClearFastStop Sequence.

The plot shows that when the servo is reconnected to the CNC at point C there is a small instantaneous error generated in the opposite direction. This is due to the 0.001 inch error remaining at reconnect time. Since the axis information available to stop.c is only updated every SYNC pass, it is not possible to exactly eliminate the CNC error. If the in-position zone were too small then the voltage moving the servos would move them across the zone before the code in stop.c could be executed again. On the other hand, if the voltage driving the servos is too small then there would be an unreasonable delay in eliminating the error. The voltage sent to the servos to eliminate the CNC error is equivalent to a feedrate of about 5 in/min.

### The Supervision Subroutines

The present set of supervision subroutines, with the exception of the fast stopping routine and the routine to change the feedrate override, all initiate a data transfer to the MCP. A consideration that needs to be studied, therefore, is how much time is taken up by the communication protocol before the supervisory computer can resume a given monitoring or control algorithm. In Chapter 4 it was seen that the ACKNOWLEDGE bit is set and cleared by the supervisory computer for each byte of data transferred. It is possible, therefore, to monitor the activity of this bit as a measure of the time required to issue a given command to the MCP.

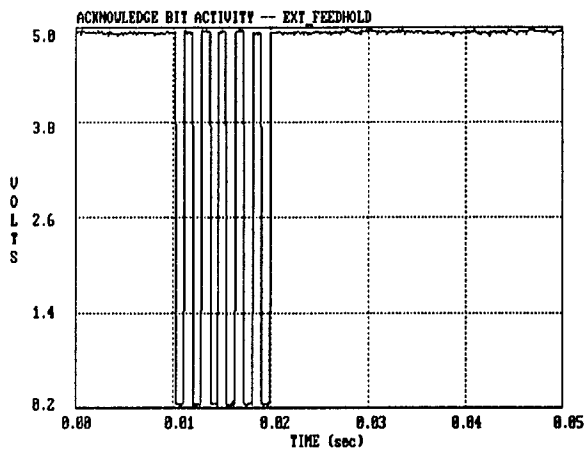
Figure 6.4 shows plots of the ACKNOWLEDGE bit activity during the execution of several of the supervision subroutines. The logic voltages were measured at the IOQ board in the MCP card cage, and in each case the data acquisition was triggered on the FIB interrupt. Several interesting features can be seen in the plots.

Figure 6.4.a shows the ACKNOWLEDGE bit activity for an ExtFeedhold. This subroutine writes a total of 6 bytes to the MCP: a short integer representing the command word; a short integer representing the type of input data (NULL); a short integer representing the type of output data (NULL). It can be seen in the plot that there are 6 logic transitions of the

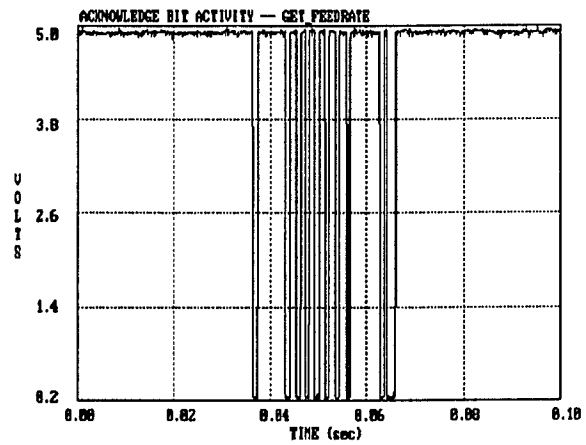
ACKNOWLEDGE bit. The time required for a one byte transfer is approximately 0.002 seconds.

Considering all the plots, it is evident that there is some variability in the amount of time it takes the MCP to begin the data transfer. This is because the interface is running in OEM\_MAIN. If the task scheduler in the MCP has tasks of a higher priority than MAIN to be executed, then they will be completed first. This is evident in Figure 6.4.b, which shows the activity for the GetFeedrate subroutine. This routine returns a long integer representing the current feedrate from the MCP. A delay of about 0.038 seconds can be seen at the start of the transfer. It can also be seen that some of the ACKNOWLEDGE bit transitions take longer than others. This is also due to the task scheduling in the MCP. In general, however, it was observed during the development of the supervision system that the subroutines completed execution in an average time of about 0.060 seconds.

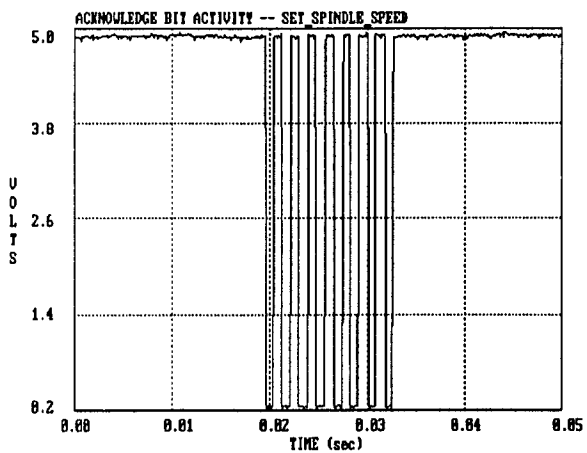
The only supervision subroutine that must be executed in a real time control loop is the SetFeedOvrd routine. The Adaptive Control scheme uses this subroutine to vary the feedrate based on the sensed vibration of the cutting process. As mentioned in Chapter 4, the subroutine was designed to complete one feedrate override command to the MCP in 0.005 seconds. This time is short compared to the overall time step of the adaptive loop, and will be shown in the next section not to have affected the stability of the control scheme.



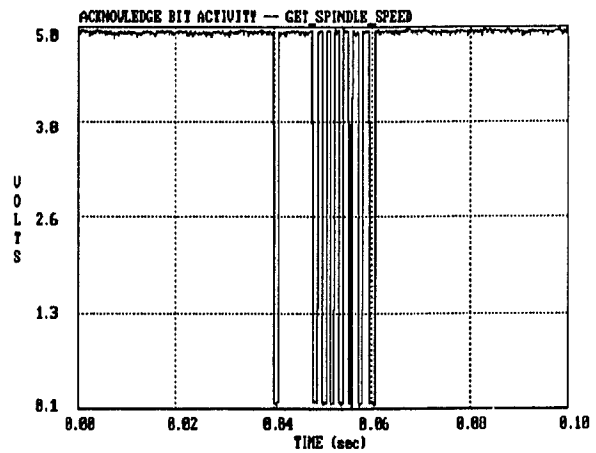
(a)



(b)



(c)



(d)

Figure 6.4. Acknowledge Bit Activity During Execution of the Supervision Subroutines.  
 a) ExtFeedhold; b) GetFeedrate;  
 c) SetSpindleSpeed; d) GetSpindleSpeed.

Figure 6.5 shows a plot of the Y axis tachometer voltage sampled during a moderate feedrate override change from 20 to 70 percent at a commanded feed of 100 in/min. The data acquisition was triggered on the FIB interrupt. Although the new feedrate override is set in the MCP at the time of the FIB interrupt, it can be seen that approximately 0.35 seconds are required to complete the feedrate change. The execution time, of course, varies depending on the size of the change in the feedrate, and is governed by the acceleration ramp tables in the MCP. The maximum feedrate override that can be commanded is 200 percent, and the smallest is 1 percent. A value of 0 percent tells the MCP to ignore the feedrate override change, and for this reason the SetFeedOvrdr subroutine cannot be used to completely stop the motion of the drives.

Figure 6.6 shows a plot of the spindle speed during a call to the SetSpindleSpeed subroutine. The speed was changed from 600 RPM to 2400 RPM. Compared to the subroutine execution time shown in Figure 6.4.c, it can be seen that a considerable amount of time can be required for the spindle to reach its new speed when a large change is commanded.

The discrete nature of the signal is due to the fact that the speed was measured by the digital tachometer used by the Spindle Torque Overload scheme. The speed values were written as voltages on a D/A channel of the data acquisition board in the supervisory computer, and were sampled by the off-line machine evaluation computer.

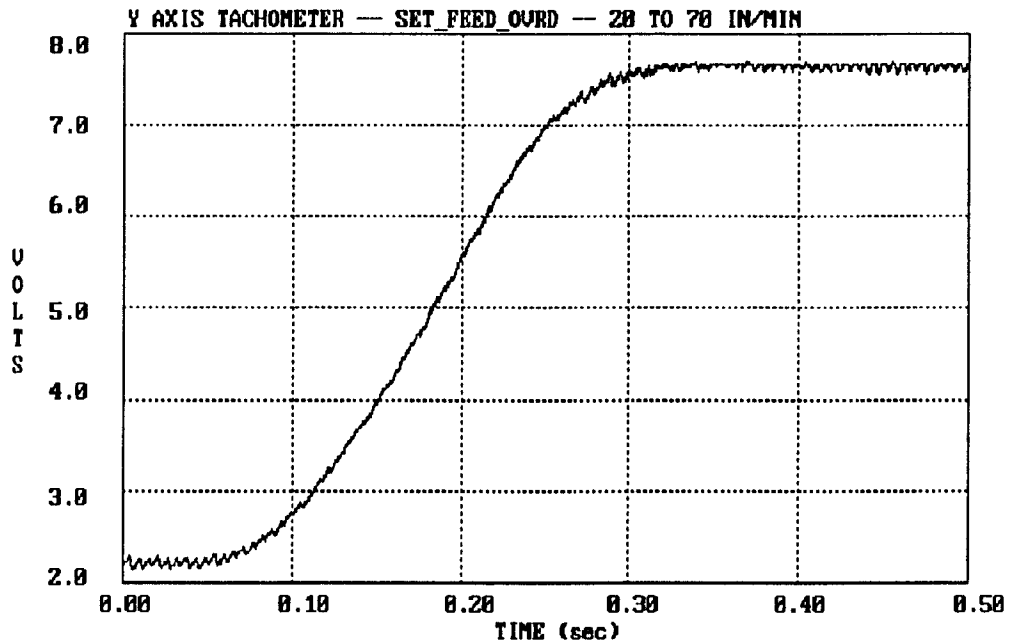


Figure 6.5. Y Axis Tachometer Voltage During a Change in Feedrate Override from 20 percent to 70 percent.

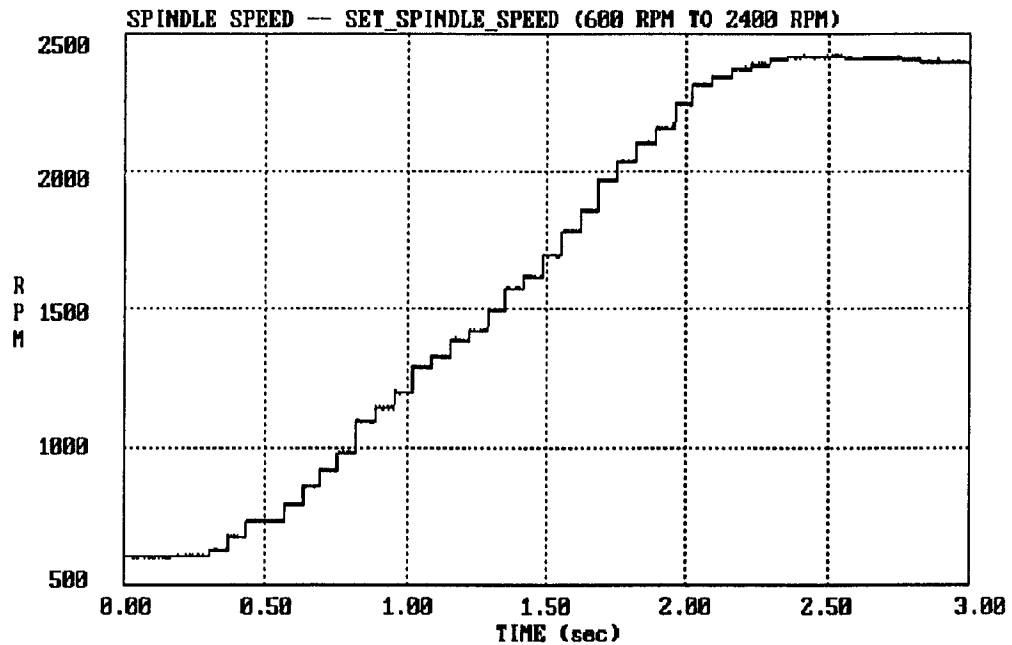


Figure 6.6. Spindle Speed During a Change from 600 RPM to 2400 RPM.

### Adaptive Control System

The Adaptive Control (A/C) system was outlined in Chapter 3. Figure 6.7 shows a block diagram of the system, as it was originally implemented by Tyler (1989) using force feedback.

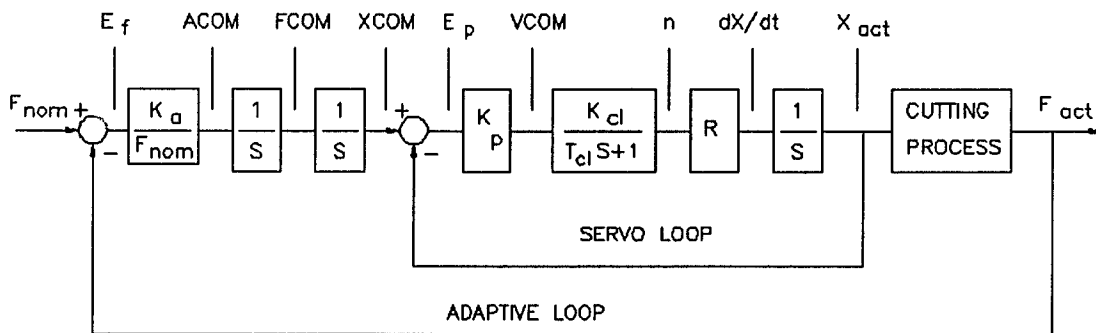


Figure 6.7. Block Diagram of the Adaptive Control System (After Tyler, 1989).

In order to integrate the A/C scheme into the supervision system, the control software was rewritten in Quick-Basic, and the supervision subroutine library was used in place of reading and writing voltages using a data acquisition board. Also, the vibration signal from the inductance probes was used in place of the original dynamometer signal, and the spindle encoder was used to synchronize the data sampling to the tooth period of the cutting tool. These software and hardware changes, which are described in the report by Wells (1991c), have essentially produced an entirely new A/C system whose performance is yet to be fully assessed.

Figure 6.8 shows the output from the inductance probes when the spindle is rotating at 600 rpm without cutting. The profile in the plot is the runout of the spindle over two revolutions. When a tool is cutting, the vibration signal is superimposed on the runout, and the runout must be subtracted from the vibration signal in the control software, tooth period by tooth period, in order to have a signal that shows only the vibration due to the cutting process. This operation was already performed in the Tool Breakage Detection scheme, by averaging 10 rotations of runout data and subtracting the average runout during the machining cuts. This method was adapted to the A/C scheme, and the computation was found not to significantly increase the time step in the adaptive loop.

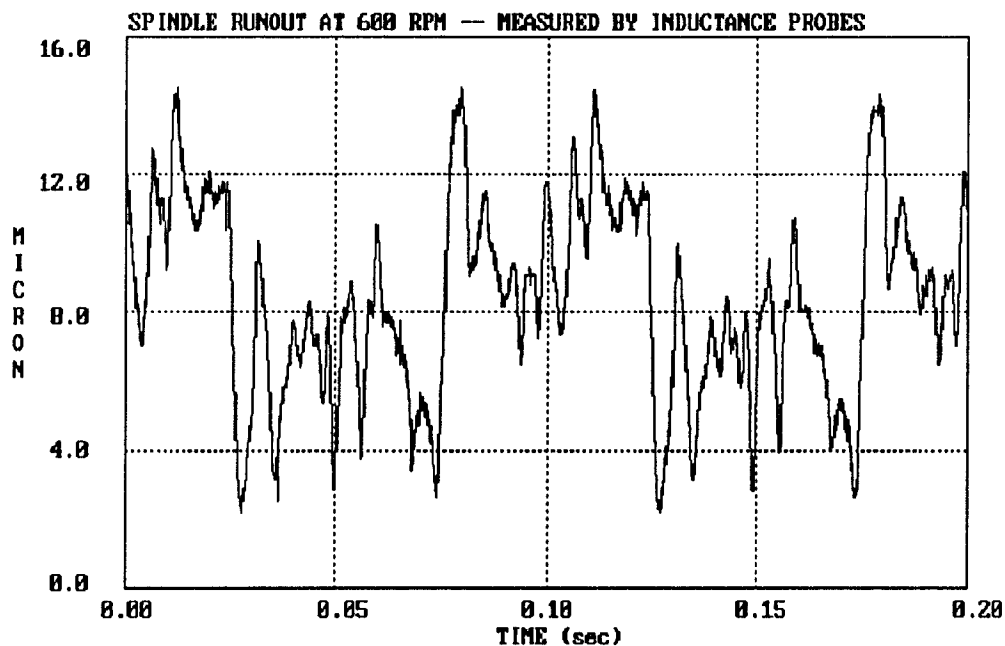


Figure 6.8. Spindle Runout Across Two Revolutions at 600 RPM Measured by the Inductance Probes.



A new data acquisition strategy was designed to take advantage of the increased resolution of the spindle encoder signal. DMA data transfer was used in the autoinitialize mode to create a circular data buffer 240 values long (see Chapter 5 for a discussion of DMA data acquisition). The buffer fills continually with data from the inductance probes, with the acquisition being clocked by the 240 pulse-per-revolution signal from the encoder. When the algorithm needs to read data, it only has to sample from the DMA buffer. The circular buffer also makes it easy to look back one tooth period, so that the change in vibration, from tooth period to tooth period, can be monitored. A diagram of the circular buffer strategy is shown in Figure 6.9.

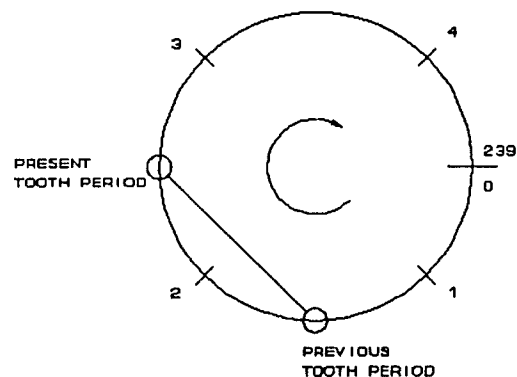


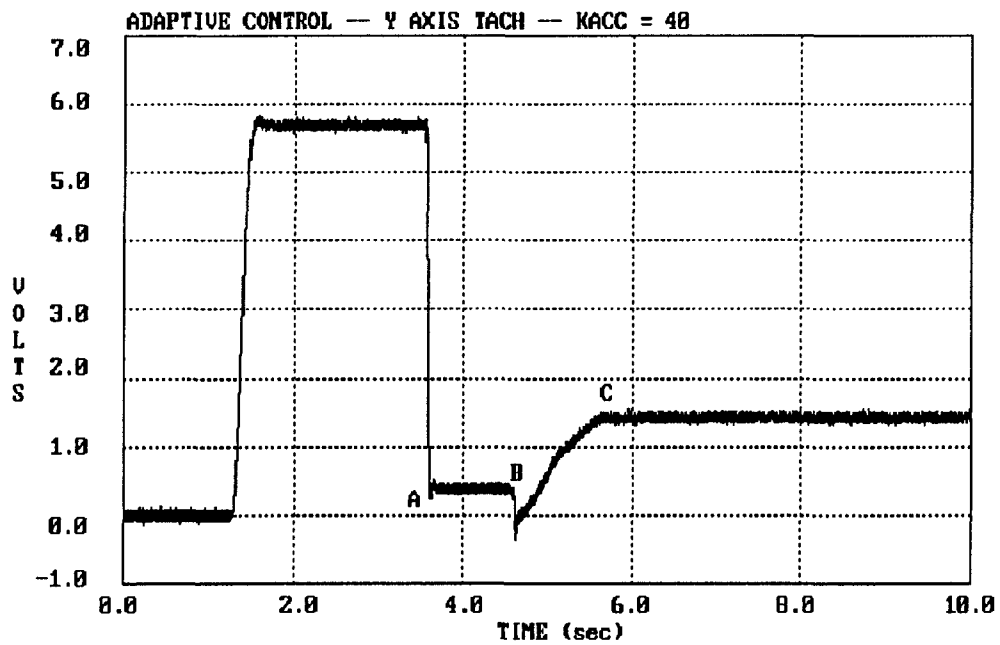
Figure 6.9. Diagram of the Circular DMA Buffer Used to Sample the Vibration Signal for a 4 Toothed Cutter.

Since two numerical integrations must be performed in order to obtain the feedrate command, it is essential to have an accurate time step for the adaptive loop. The A/C program was also modified to obtain the time step automatically, so

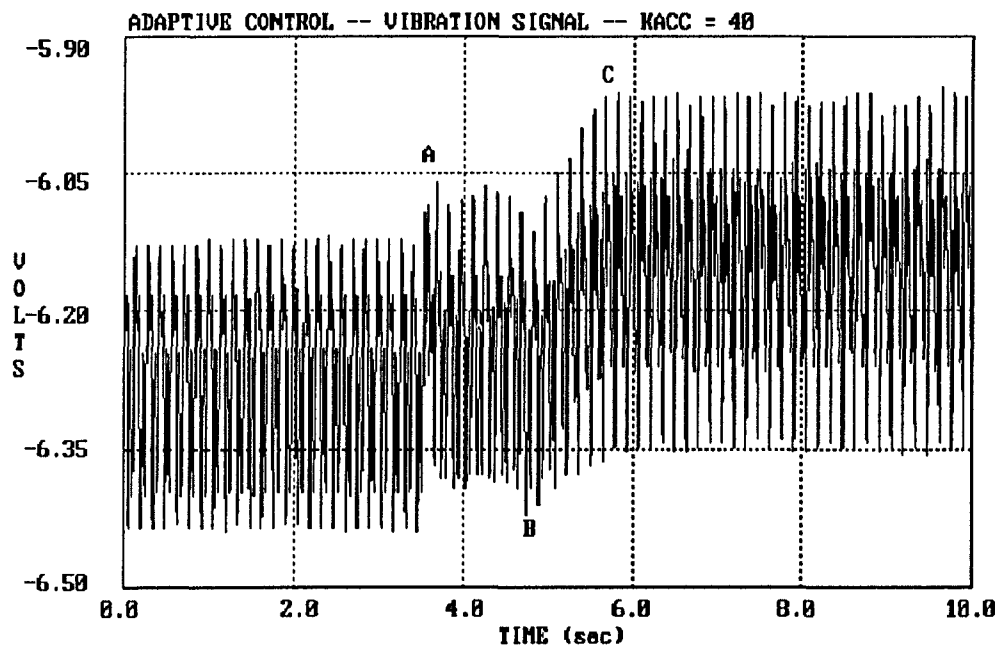
that the scheme could be run on computers with different processing speeds. At the start of the program the adaptive loop is run off-line for 300 cycles, and the execution time is measured using the timer in the computer. The net time step for the adaptive loop on the present supervisory computer was computed to be 0.007 seconds, which equates to 5 data samples per tooth period at the spindle speed used in the tests.

For the cutting tests, a 0.75 inch diameter by 4.25 inch long 4 fluted HSS end mill was used to cut 7075-T6 aluminum. The spindle speed was 420 RPM, and the maximum cutting feedrate was 13.2 in/min. This gave a feed per tooth of 0.008 inches. An 0.150 inch axial depth of cut was used for up milling at 25 percent radial immersion. The cuts were made in the negative Y axis direction. The feedrate in the transient loop was four times the cutting feedrate (52.8 in/min). These parameters are the same as those used by Tyler (1989).

Figure 6.10.a shows a plot of the Y axis tachometer signal during a representative cut with the value of the adaptive gain,  $K_a$ , set at 40. Since outputting the processed inductance probe signal from the A/C program during the adaptive loop is computationally expensive, the voltage directly from the inductance probes, without the runout subtracted, was sampled during the cut. The signal is shown in Figure 6.10.b. In order to bring out features in the signal, it was passed through a 26 Hz low pass filter to remove the tooth frequency (28 Hz) and its higher harmonics.



(a)



(b)

Figure 6.10 Demonstration of the Adaptive Control System.  
 a) Y Axis Tachometer Signal.  
 b) Vibration Signal from the Inductance Probes.

When the impact of the tool and the workpiece was detected by the scheme, point A on the plots, the fast stop was triggered. The axis velocity dropped to zero in about 0.035 seconds. The ClearFastStop subroutine was called immediately, and the CNC error was eliminated at point B. The adaptive loop in the scheme then took over and proceeded with the cut, increasing the feedrate until the nominal cutting feedrate was reached at point C.

The test cuts of the new A/C scheme showed that the vibration signal from the inductance probes, instead of a force signal from a dynamometer, could be used to regulate the cutting process if the spindle runout were subtracted for each tooth period. It was also observed that the adaptive loop performed well for values of  $K_a$ , from 20 to 80. However, more tests need to be performed on the A/C system to fully explore the vibration signal, and ways to process it to extract information about the cutting process.

#### Tool Breakage Detection System

The Tool Breakage Detection system (Tarng, 1988; Vierck, 1991) is able to recognize when an insert on a face mill, or a flute on an end mill, is broken. As part of the on-line supervision system, the scheme is designed to call the fast stopping subroutine when a broken tooth is detected during a cut.

Figure 6.11 shows a plot of the first difference and average displacement values measured during a cut with an undamaged tool. The cut was made on cast iron using an eight toothed 4.25 inch diameter face mill on a #50 taper V-flange tool holder (No. CV50SM200240). The inductance probes were used to sense the vibration, and the Tool Breakage Detection program by Vierck (1991) was used to sample and plot the data. For Figure 6.12, another cut was made under the same conditions with one of the inserts removed from the face mill. This simulated tool breakage.

The entry and exit to the cuts can be clearly seen in the plots of the average displacement. As described in Chapter 3, the signature of tooth breakage is evident in the first difference values of the broken cutter (Figure 6.12). The vibration amplitudes, both positive and negative, are much greater for the damaged cutter. This violates the upper and lower trigger thresholds, and causes the fast stop command to be issued to the MCP.

Since it is difficult to simulate tool breakage in the middle of a cut, a plot of the fast stop being triggered by the scheme has not been included. When a tool with one insert removed enters the cut, the scheme reacts immediately. The data would show only the runout of the tool and the brief vibration at the moment of entry. Moreover, the data from both machining passes clearly show that the first difference values give an excellent indication of tool breakage.

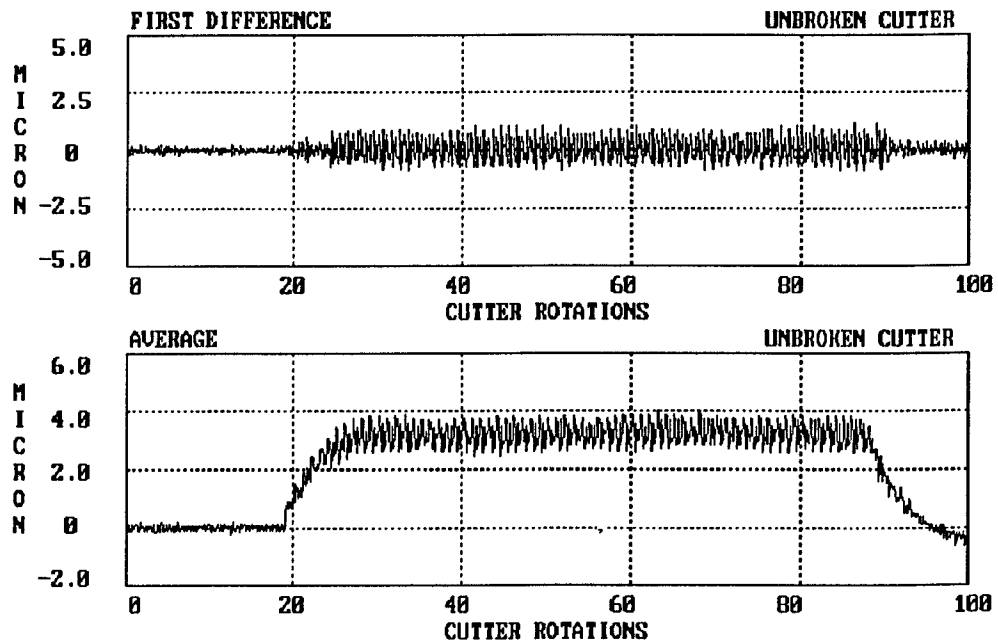


Figure 6.11. First Difference and Average Displacement Values for an Undamaged Cutter.

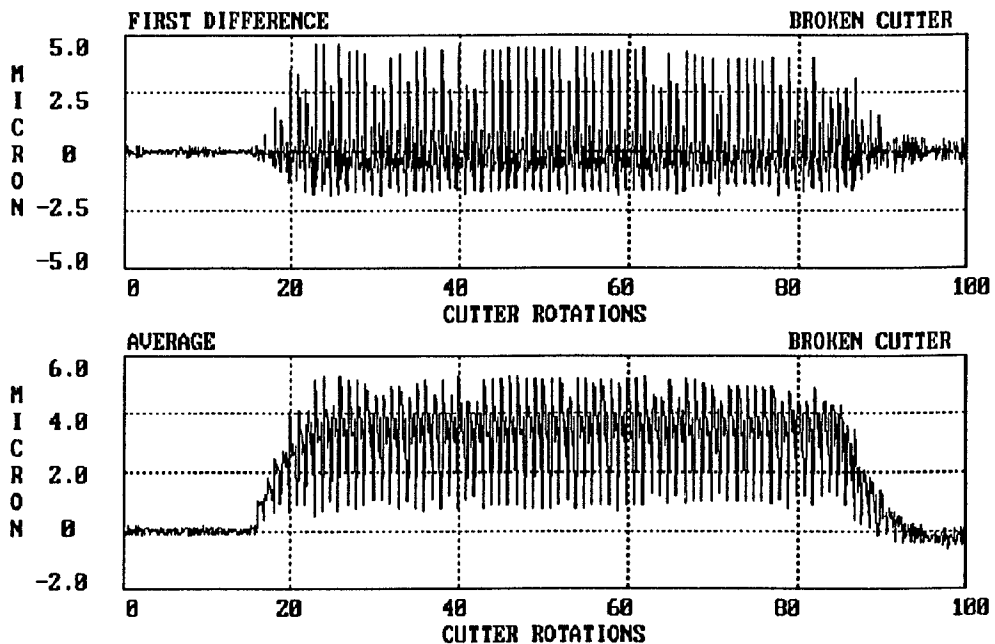


Figure 6.12. First Difference and Average Displacement Values for a Damaged Cutter.

Chatter Recognition and Control System

In order to demonstrate the Chatter Recognition and Control scheme as part of the on-line supervision system on the Omnimil, machining cuts were made on cast iron using an 8 toothed 4.25 inch diameter face mill on a #50 taper V-flange long extension tool holder (No. C50-15SM600) with Silicon Nitride inserts. The Transfer Function Module of PCDATA was used to measure the TF of the tool. Figure 6.13 shows the Transfer Function in the X axis direction. It can be seen that the tool has modes of interest at 251 Hz and 327 Hz, with the latter mode being associated with the most negative real part of the Transfer Function.

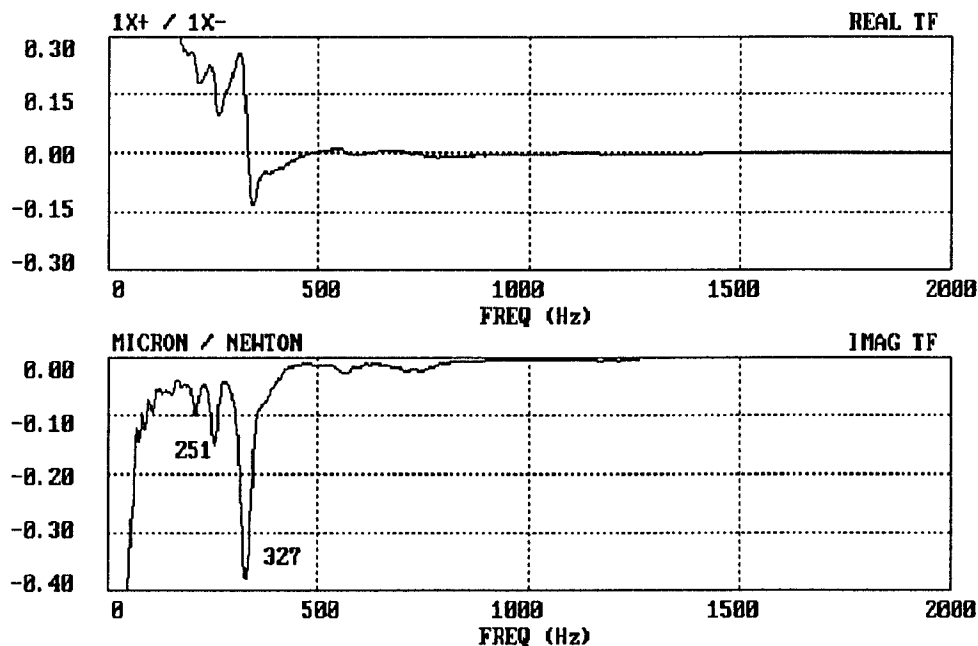


Figure 6.13. Transfer Function of the Tool Used in the Chatter Recognition and Control Tests.

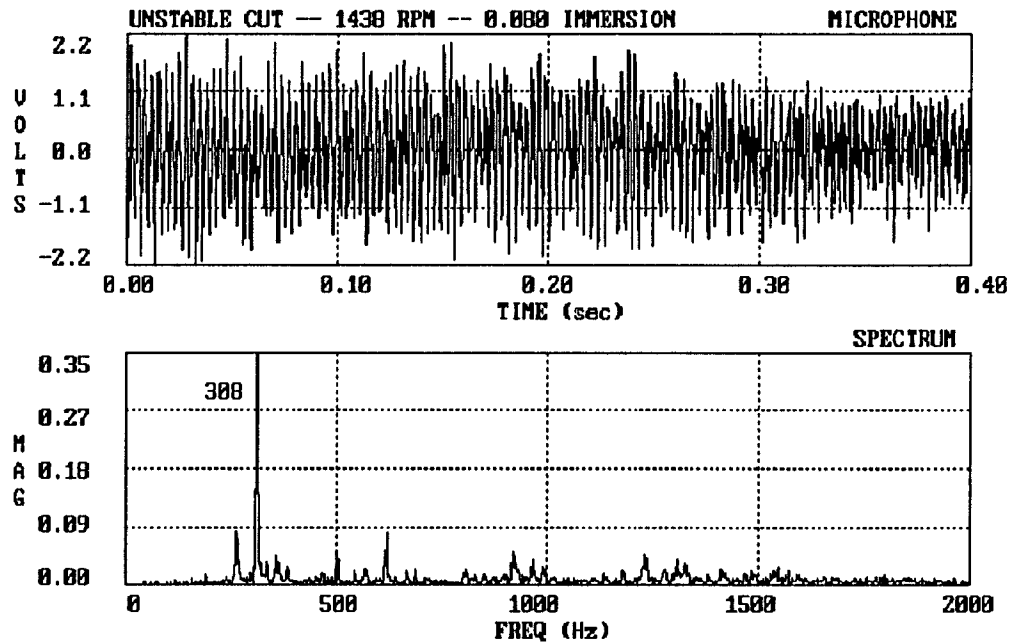
Using a feedrate of 80 in/min, and a commanded spindle speed of 1450 RPM, which produced an actual speed of 1438 RPM, several test cuts were made. The value of  $b_{lim}$  for the tool was found to be just above 0.070 inches. For the demonstration, an unstable axial immersion of 0.080 was chosen.

The cut was immediately very unstable, with the chatter sound being quite pronounced. Upon detection of chatter, the system called the FastStop subroutine. The spindle speed was adjusted, using the SetSpindleSpeed subroutine, according to the algorithm outlined in Chapters 3 and 5. The new commanded speed was 2429 RPM, which produced an actual spindle speed of 2394 RPM. The feedrate was adjusted to 134.4 in/min by the SetFeedOvrdr subroutine so that a constant feed per tooth was maintained. The fast stop was then cleared, and the programmed motion continued. The remainder of the cut was stable.

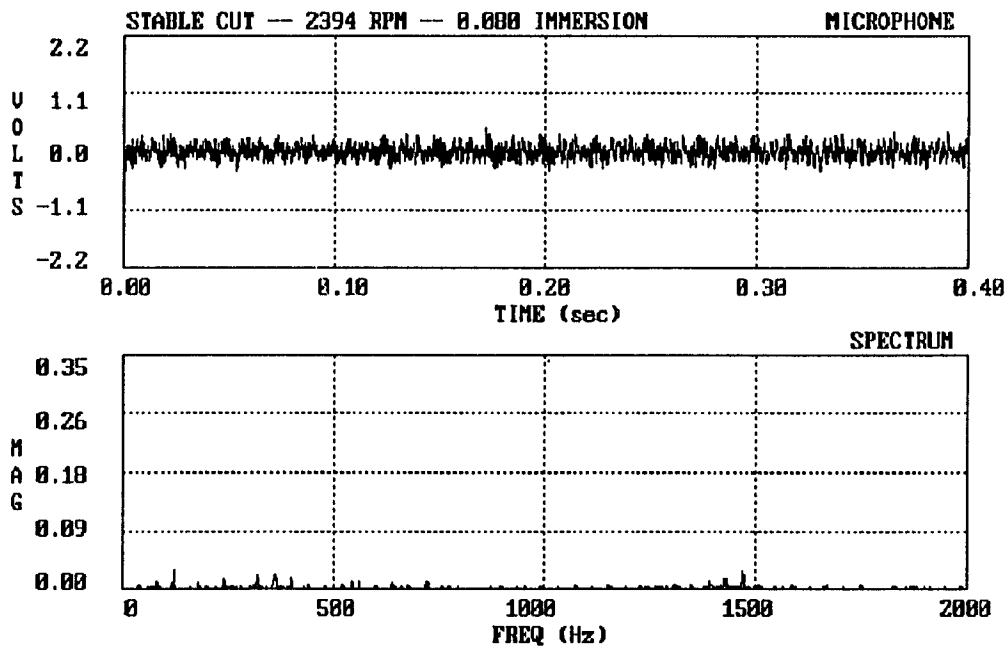
Figure 6.14.a shows the microphone data sampled during the unstable cut, and its spectrum. The chatter peak is located at 308 Hz, which corresponds to the 327 Hz mode shown in Figure 6.13. The chatter frequency appears to be lower than the modal frequency shown in the Transfer Function because the load on the tool due to the cutting force most likely shifted the modal frequencies lower during the cut.

Figure 6.14.b shows the microphone data and spectrum of the stable cut produced by the automatic spindle speed regulation. For purposes of comparison, the microphone data and spectrum have been plotted at the same scale as Figure





(a)



(b)

Figure 6.14. Microphone Data Sampled During the Demonstration of the Chatter Recognition and Control System.  
 a) Initial Unstable Cut;  
 b) Stable Cut after Spindle Speed Regulation.

6.14.a. The peaks in the spectrum now belong to the harmonics of the spindle frequency, 39.9 Hz. The system has produced a stable cut resulting in a significant increase in the MRR.

The increase in spindle speed produced by the automatic speed regulation resulted in a 66 percent increase in the MRR. A series of cuts were then made at the stable speed found by the system, and it was determined that the axial depth of cut could be increased to 0.110 inches before chatter again appeared. This represents another 57 percent increase in the MRR compared to the stable 0.070 inch immersion cut made at the original speed.

It should be noted that the speed range of the Omnihil restricts application of the chatter system to tools with relatively low natural frequencies. For tools with high natural frequencies, such as end mills, a high speed spindle is needed to reach the stable speeds selected by the scheme.

#### Spindle Torque Overload System

In Chapter 3 it was shown that, although armature current is the best indicator of motor torque, the speed drop (deceleration) of the spindle could be taken as an indication of the inability of the motor to recover from a sudden overload. The Spindle Torque Overload program, listed in Appendix B, monitors the speed by processing the once-per-revolution signal from the spindle encoder. If the actual

speed falls below the commanded CNC speed by a specified limit, then a torque overload is declared and a fast stop is commanded to the MCP. If a speed change is being commanded from a CNC block, or by one of the supervision schemes, the routine resets the limit based on the new speed and continues monitoring.

Figure 6.15 shows a plot of the spindle speed, and Figure 6.16 shows the Y axis tachometer voltage, during a spindle slow down while cutting cast iron with a 4.25 inch diameter 8 toothed face mill on a long extension. The feedrate was 80 in/min in a full slotting cut with an axial immersion of 0.050 inches. The CNC commanded speed was 1450 RPM (1438 RPM actual).

Cuts using these parameters had been monitored with the Spindle Torque Overload software, and it was determined that the spindle speed dropped to about 1409 RPM at the entry to the cut. For the purposes of the demonstration the limit speed was set to 98 percent of the CNC speed, although a value of 95 percent would be more reasonable in an actual monitoring situation.

The discrete nature of the spindle speed signal shown in Figure 6.15 is a result of the digital tachometer implemented in the supervision computer, and is also due to the small speed variation used in the demonstration. When the speed dropped below the limit, the fast stopping routine was called and the spindle recovered to its non-cutting speed. Figure

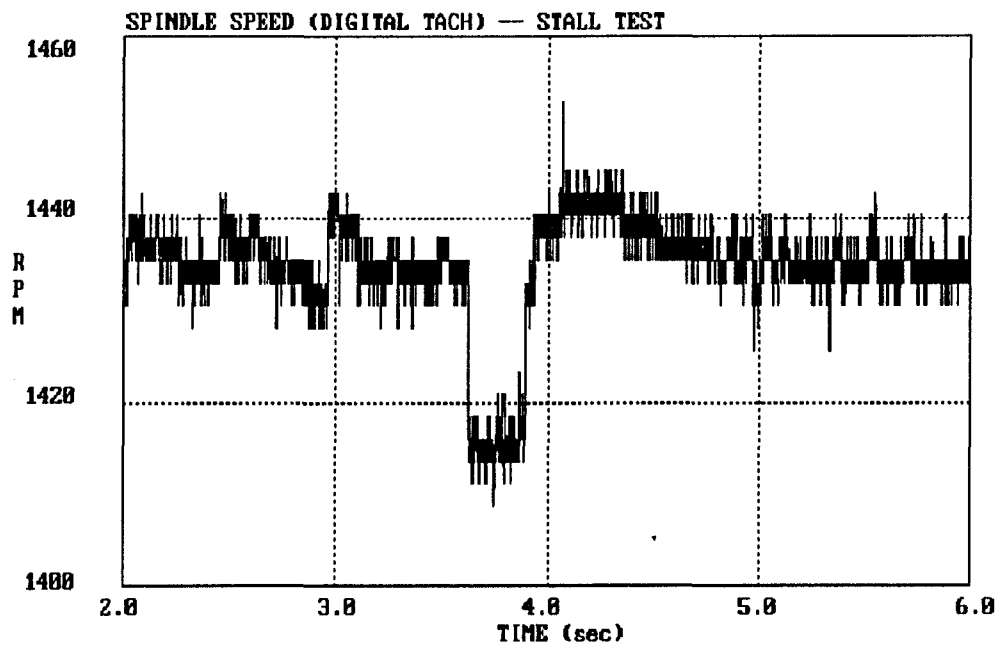


Figure 6.16. Spindle Speed During a Slow Down at the Entry to a Cut.

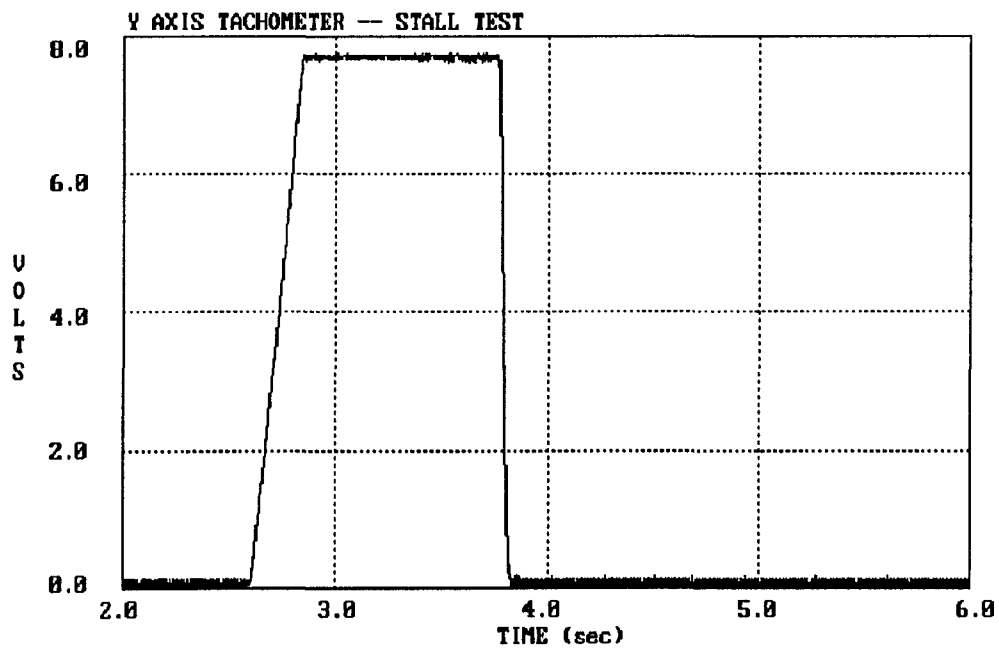


Figure 6.17. Y Axis Tachometer Showing the Fast Stop Called by the Spindle Slow Down.

6.16 shows the beginning of the motion command to the Y axis, and the fast stop being triggered by the spindle slow down.

The 0.100 second delay evident between the speed drop and the fast stop is due to the software overhead required to update the tachometer and to check that the speed change did not originate in the CNC (using the GetSpindleSpeed subroutine). Although the fast stop would be triggered before the spindle speed approached zero in a true stall, this delay may not be acceptable at higher speeds and feeds. It is proposed, therefore, that the scheme could eventually be developed to make use of the spindle motor current to sense torque directly.

The demonstrations and experimental results described in this chapter show that the individual monitoring and control schemes have been successfully integrated into a machine tool supervision system. In Chapter 7 the research will be summarized, and suggestions for further developing the system, and the individual monitoring and control schemes, will be presented.

## CHAPTER 7 CONCLUSIONS AND RECOMMENDATIONS

The experimental results presented in Chapter 6 show that each of the monitoring and control schemes in the on-line supervision system performed as required. The supervision subroutines, described in Chapter 4, provided an effective means for the schemes to take control of the Omnimil when sensed data indicated that action must be taken.

The A/C system successfully regulated the cutting feedrate by monitoring the spindle vibration. However, more cutting tests are needed to fully develop the new implementation of the scheme, especially as regards the processing of the inductance probe signal.

The Broken Tool Detection system was shown to be able to identify the characteristic signature of tool breakage. Vierck (1991) is presently developing the scheme to include automatic thresholding, and Walters (1991) is studying the effect of different signal processing techniques.

The Chatter Regulation and Control system demonstrated that stable cutting speeds could be automatically selected once chatter was detected. The speed range of the Omnimil makes the scheme most successful when tools with relatively low natural frequencies are used.

The Spindle Torque Overload system was able to protect against a stall of the spindle motor by commanding a fast stop when the actual spindle speed differed from the CNC speed by a specified limit. The scheme could be improved, however, by using the spindle motor current as an indication of torque, instead of sensing a speed change.

The supervision subroutines were shown to provide the machine control necessary for the supervision schemes to operate effectively. An improvement to the fast stopping routine would be updating the axis position tables inside the MCP to eliminate the CNC following error, instead of moving the drives. This will be studied in future research.

The overall performance of the supervision system at this stage may be characterized as satisfactory, but it must be noted that the system is still in its infancy. Real improvements will be identified only after thorough testing of each of the schemes in a variety of cutting situations.

In order for the supervision system to be truly comprehensive, the monitoring and control schemes must be run simultaneously. This could be accomplished by either parallel processors or distributed computing. The parallel processors could be four DSP chips, located in the supervisory computer, each of which is executing one of the schemes. The use of a network of distributed computers to execute the schemes in parallel is presently being studied by Walters (1991).

The off-line machine evaluation program, PCDATA, was described in Chapter 5. The Data Acquisition Module was used to monitor sensor signals throughout the experimental work, and the Transfer Function Module was used to measure Transfer Functions of tools used in cutting the tests.

The off-line system could be significantly enhanced by adding modal parameter estimation and curve fitting. This facility, operating on Transfer Function data from PCDATA, could then be used to automatically generate input to the milling simulation programs that have been developed in the Machine Tool Laboratory. The result could be a very powerful automated machine evaluation system combining actual measurements with simulations of the cutting process.

An implicit objective in this research has been to demonstrate the feasibility of having an external supervisory computer work in conjunction with a machine tool controller to monitor and control the metal cutting process. The success of the research suggests that manufacturers of machine tool controllers should begin to provide direct support for communication between the CNC and external computers. Also, they should make available, through the interface, the critical CNC control parameters necessary for machine tool users to implement various monitoring and control strategies. This can lead to a higher level of productivity and product quality in the manufacturing culture of this country.



APPENDIX A  
LISTING OF THE SUPERVISION SOFTWARE

This appendix includes listings of the interface and supervision programs. It should be noted that global variables and flags for `iface.c` and `stop.c` are declared in the header file `oem_dec.h`, which is not included in the listing.

Supervision Computer Interface

```

/*****
/*   Title: PCIFACE.C ver 1.0
/*   By:    R.L. Wells and R. Walters
/*   Date:  05-31-91
/*
/* Interface between the supervision computer and the
/* FlexMate CNC. The supervision subroutines are:
/* void Dt2817Init (void);
/* void FastStop (void);
/* void ClearFastStop (void);
/* void ExtFeedhold (void);
/* void SetFeedOvrd (int pfp);
/* void GetFeedrate (long *fr);
/* void GetManFeedOvrd (int *mfp);
/* void SetSpindleSpeed (int ns);
/* void GetSpindleSpeed (int *rpm);
*****/

#include <stdio.h>
#include <math.h>
#include <conio.h>
#include <time.h>

/* Define the control and data registers for the DT-2817. */
```

```
#define CONTROL_REGISTER 0x228
#define DATA_REG0      0x229
#define DATA_REG1      0x22a
#define DATA_REG2      0x22b
#define DATA_REG3      0x22c

/* Assign the working bit values. */

#define FIB_INTERRUPT    0x80
#define TRANS_REQ        0x04
#define ACKNOWLEDGE      0x01

/* Define command data type constants. */

#define RPC_SHORT        0
#define RPC_LONG         1
#define RPC_SHORT_ARY    2
#define RPC_LONG_ARY     3
#define RPC_VOID         4

/* The command name numbers MUST MATCH the command name */
/* numbers that are defined in the FlexMate interface */
/* program IFACE.C. They MUST be between 200 and 250! */

#define EXT_FEED_HOLD    201
#define SPEED_CHANGE     202
#define GET_SPEED        203
#define GET_FEED         204
#define GET_FEED_OVRD    205

/* Variables used for input and output data are global. */

unsigned int dataout0, dataout1, datain2, datain3;

/* Function prototypes for the low level interface. */

void interrupt_mcp (int int_cmd);
void write_to_mcp (int output_data_value);
void read_from_mcp (int *input_data_value);
void mcp_call (int mcp_cmd, int out_data_type, char *out_data,
int in_data_type, char *in_data);
void acknowledge (void);

/* Function prototypes for the supervision routines */

void Dt2817Init (void);
void FastStop (void);
void ClearFastStop (void);
void ExtFeedhold (void);
void SetFeedOvrD (int pfp);
void GetFeedrate (long *fr);
void GetManFeedOvrD (int *mfp);
```

```

void SetSpindleSpeed (int ns);
void GetSpindleSpeed (int *rpm);

/* ----- */

void FastStop()
{
/* Execute the FAST STOP routine. */
    interrupt_mcp (0xfd);
    acknowledge ();
} /* end of FastStop */

/* ----- */

void ClearFastStop()
{
/* Clear the FAST STOP routine. */
    interrupt_mcp (0xfc);
    acknowledge ();
} /* end of ClearFastStop */

/* ----- */

void ExtFeedhold()
{
/* Each call to this function TOGGLES an EXTERNAL FEEDHOLD. */
mcp_call (EXT_FEED_HOLD, RPC_VOID, NULL, RPC_VOID, NULL);
} /* end of ExtFeedhold */

/* ----- */

void GetFeedrate (long *fr)
{
/* Get the current FEED RATE (in/min*1000) from the MCP. */
mcp_call (GET_FEED, RPC_VOID, NULL, RPC_LONG, (char *)fr);
*fr /= 1000;
} /* end of GetFeedrate */

/* ----- */

void GetManFeedOvrd (int *mfp)
{
/* Get the MANUAL FEED OVERRIDE (percent) from the MCP. */
mcp_call (GET_FEED_OVRD, RPC_VOID, NULL, RPC_SHORT, (char *)mfp);
} /* end of GetManFeedOvrd */

/* ----- */

void GetSpindleSpeed (int *rpm)
{
/* Get the current SPINDLE SPEED * OVERRIDE from the MCP. */
mcp_call (GET_SPEED, RPC_VOID, NULL, RPC_SHORT, (char *)rpm);
}

```

```

} /* end of GetSpindleSpeed */

/* ----- */

void SetSpindleSpeed (int ns)
{
/* Set the SPINDLE SPEED (RPM) in the MCP. */
mcp_call (SPEED_CHANGE, RPC_SHORT, (char *)&ns, RPC_VOID, NULL);
} /* end of SetSpindleSpeed */

/* ----- */

void SetFeedOvr (int pfp)
{
int i = 0;
/* Set the FEEDRATE OVERRIDE (percent) in the MCP. */

outp (DATA_REG0, ~(pfp));
dataout1 = inp (DATA_REG1) | FIB_INTERRUPT;

/* Hold FlexMate FIB interrupt bit high for 5 ms. */
/* NOTE: The speed of the for loop depends on the */
/* clock speed of the supervisory computer! */

for (i = 0; i < 300; i++) outp (DATA_REG1, dataout1);

/* Clear the FlexMate FIB interrupt bit. */
dataout1 = inp (DATA_REG1) ^ FIB_INTERRUPT;
outp (DATA_REG1, dataout1);
/* Clear the interrupt command value. */
outp (DATA_REG0, ~(0x00));

} /* end of SetFeedOvr */

/* ----- */

/* ***** */
/* The functions below this comment constitute the low level */
/* interface and should not be changed WITHOUT GOOD REASON! */
/* ***** */

void Dt2817Init ()
{
/* Set ports 0 & 1 for output and ports 2 & 3 for input. */
outp (CONTROL_REGISTER, 0x3);

/* Be sure that both output ports are low. Note that */
/* the IOQ reads inverted, and the FIB reads normal. */
outp (DATA_REG0, 0xff);
outp (DATA_REG1, 0x7f);

} /* end of Dt2817Init */

```

```

/* ----- */
/*****
/* This function handles all the communication with the MCP */
*****/

void mcp_call (int mcp_cmd, int out_data_type, char *out_data,
               int in_data_type, char *in_data)
{
  unsigned int temp;

  /* Get the attention of the FlexMate MCP. */
  interrupt_mcp (0xfe);

  /* Write the command code to the MCP. */
  write_to_mcp (mcp_cmd);

  /***** Write the output data *****/

  write_to_mcp (out_data_type);
  switch (out_data_type)
  {
    case RPC_SHORT: /* Send short integer to MCP. */
      write_to_mcp (*(int *)out_data);
      break;

    case RPC_LONG: /* Send long integer to MCP. */
      /* Send high order word */
      write_to_mcp (*(unsigned *) (out_data + 2));
      /* Send low order word */
      write_to_mcp (*(unsigned *)out_data);
      break;

    case RPC_SHORT_ARY: /* Send array of shorts to MCP. */
      break;

    case RPC_LONG_ARY: /* Send array of longs to MCP. */
      break;

    case RPC_VOID: /* Send no data to MCP */
      break;

    default:
      break;
  }

  /***** Read the input data. *****/

  write_to_mcp (in_data_type);
  switch (in_data_type)
  {
    case RPC_SHORT: /* Read short integer from MCP. */

```

```

        read_from_mcp (&temp);
        *(int *)in_data = temp;
        break;

    case RPC_LONG: /* Read long integer from MCP. */
        read_from_mcp (&temp);
        /* Store high order word */
        *(unsigned int *) (in_data + 2) = temp;
        read_from_mcp (&temp);
        /* Store low order word */
        *(unsigned int *) (in_data) = temp;
        break;

    case RPC_SHORT_ARY: /* Read array of shorts from MCP */
        break;

    case RPC_LONG_ARY: /* Read array of longs from MCP */
        break;

    case RPC_VOID: /* Read no data from MCP. */
        break;

    default:
        break;
}
outp (DATA_REG0, ~(0x00)); /* Clear the data line */
} /* end of mcp_call */

/* ----- */

void interrupt_mcp (int int_cmd)
{
/* If int_cmd = 0xfe, the interface is executed.      */
/* If int_cmd = 0xfd, the set fast stop is executed.  */
/* If int_cmd = 0xfc, the clear fast stop is executed. */

    outp (DATA_REG0, ~(int_cmd));

/* Set the FlexMate FIB interrupt bit. */
    dataout1 = inp (DATA_REG1) | FIB_INTERRUPT;
    outp (DATA_REG1, dataout1);

/* Wait for Transmit Request bit to go high. */
    while (!(inp(DATA_REG3) ^ TRANS_REQ) & TRANS_REQ);

/* Clear the FlexMate FIB interrupt bit. */
    dataout1 = inp (DATA_REG1) ^ FIB_INTERRUPT;
    outp (DATA_REG1, dataout1);
    outp (DATA_REG0, ~(0x00));
} /* end of interrupt_mcp */

```

```

/*****
/* This subroutine reads a two-byte word from the I/O space */
/* of the MCP. NOTE: The logic of the MCP I/O space is */
/* inverted, so that a 1 in the computer is a 0 in the MCP. */
*****/

void read_from_mcp (int *input_data_value)
{
  unsigned int low_byte, high_byte;

  /* Low byte transfer. */

  /* Wait for a transmit request signal from the MCP. */
  while (!((inp(DATA_REG3) ^ TRANS_REQ) & TRANS_REQ));

  /* Read the low byte from the MCP on port 2. */
  low_byte = inp (DATA_REG2) ^ 0xff;

  /* Set the ACKNOWLEDGE bit on port 1. */
  dataout1 = inp (DATA_REG1) ^ ACKNOWLEDGE;
  outp (DATA_REG1, dataout1);

  /* Wait for a transmit stop signal from the MCP. */
  while (!(inp (DATA_REG3) & TRANS_REQ));

  /* Clear the ACKNOWLEDGE bit on port 1. */
  dataout1 = inp (DATA_REG1) | ACKNOWLEDGE;
  outp (DATA_REG1, dataout1);

  /* High byte transfer. */

  /* Wait for a transmit request signal from the MCP. */
  while (!((inp(DATA_REG3) ^ TRANS_REQ) & TRANS_REQ));

  /* Read the high byte from the MCP on port 2. */
  high_byte = inp (DATA_REG2) ^ 0xff;

  /* Set the ACKNOWLEDGE bit on port 1. */
  dataout1 = inp (DATA_REG1) ^ ACKNOWLEDGE;
  outp (DATA_REG1, dataout1);

  /* Wait for a transmit stop signal from the MCP. */
  while (!(inp (DATA_REG3) & TRANS_REQ));

  /* Clear the ACKNOWLEDGE bit on port 1. */
  dataout1 = inp (DATA_REG1) | ACKNOWLEDGE;
  outp (DATA_REG1, dataout1);

  /* Construct the data word. */
  *input_data_value = low_byte + high_byte * 256;

} /* end of read_from_mcp */

```

```

/*****
/* This subroutine writes a two-byte word to the I/O space */
/* of the MCP. NOTE: The logic of the MCP I/O space is */
/* inverted, so that a 1 in the computer is a 0 in the MCP. */
/*****

void write_to_mcp (int output_data_value)
{
  unsigned int low_byte, high_byte;

  /* Obtain low and high bytes of the output_data_value word. */
  high_byte = output_data_value / 256;
  low_byte = output_data_value - high_byte * 256;

  /* Wait for a transmit request signal from the MCP. */
  while (!(inp (DATA_REG3) ^ TRANS_REQ) & TRANS_REQ));

  /* Put the low byte on port 0. */
  outp (DATA_REG0, ~(low_byte));

  /* Set the ACKNOWLEDGE bit on port 1. */
  dataout1 = inp (DATA_REG1) ^ ACKNOWLEDGE;
  outp (DATA_REG1, dataout1);

  /* Wait for a transmit stop signal from the MCP. */
  while (!(inp (DATA_REG3) & TRANS_REQ));

  /* Clear the ACKNOWLEDGE bit on port 1. */
  dataout1 = inp (DATA_REG1) | ACKNOWLEDGE;
  outp (DATA_REG1, dataout1);

  /* Wait for a transmit request signal from the MCP. */
  while (!(inp (DATA_REG3) ^ TRANS_REQ) & TRANS_REQ));

  /* Put the high byte on port 0. */
  outp (DATA_REG0, ~(high_byte));

  /* Set the ACKNOWLEDGE bit on port 1. */
  dataout1 = inp (DATA_REG1) ^ ACKNOWLEDGE;
  outp (DATA_REG1, dataout1);

  /* Wait for a transmit stop signal from the MCP. */
  while (!(inp (DATA_REG3) & TRANS_REQ));

  /* Clear the ACKNOWLEDGE bit on port 1. */
  dataout1 = inp (DATA_REG1) | ACKNOWLEDGE;
  outp (DATA_REG1, dataout1);

} /* end of write_to_mcp */

/* ----- */

```



```

void acknowledge ()
{
/*****
/* This subroutine can be used to acknowledge an interrupt */
/* to the MCP without having to go through a data transfer. */
/*****

/* Set the ACKNOWLEDGE bit on port 1. */
    dataout1 = inp (DATA_REG1) ^ ACKNOWLEDGE;
    outp (DATA_REG1, dataout1);

/* Wait for a transmit stop signal from the MCP. */
    while (!(inp (DATA_REG3) & TRANS_REQ));

/* Clear the ACKNOWLEDGE bit on port 1. */
    dataout1 = inp (DATA_REG1) | ACKNOWLEDGE;
    outp (DATA_REG1, dataout1);

} /* end of acknowledge */

/* ----- */

```

### FlexMate Motion Co-Processor Interface

```

/*****
/* Title:    IFACE.C (Interface handler for Omnimil machine */
/*           supervision system, Univ. of Florida) */
/* Date:    05-31-91 */
/* Authors: R. Walters and R.L. Wells */
/* Version: 1.0 */
/*****

#include <mcl.h>
#include <globals.h>
#define IFACE
#include "oem_dec.h"
#define fib_port_address 0x90

/* Add new supervision command numbers here (201-250) */

#define ext_feedhold    201
#define speed_change    202
#define get_speed       203
#define get_feed        204
#define get_feed_ovrd   205

```

```

enum
{
    rpc_short,
    rpc_long,
    rpc_short_ary,
    rpc_long_ary,
    rpc_void
} data_type;

/* Function prototypes */

int  read_word();
int  write_word();
void iface_set_bit();
void iface_clear_bit();
void acknowledge();

int  hold, error_code, rpc_ptr;

/* ----- */

void iface_init() /* This routine called once at power on time */
{
    /* Clear the output lines. */
    output_data_address = 0x00;
    unpack(output_data_address, SCT + output_port_address, 16);
    outport(output_port_address, output_data_address);

    /* Set all control flags and variables to false */
    int_cmd = 0;
    fast_stop_flag = 0;
    fast_stop_state = 0;
    feed_hold_flag = 0;
    pfo_percent = 0;

    /* Set maximum feed override to 201 percent */
    getw(FROTOP) = 0x00c9;

    /* Initialize the FIB interrupt (FIB bit zero) */

    /* set FIB card for address 60 */
    getw(FASTIOTB) = 0x60;
    /* get maps to OEM_MAIN */
    getw(FIMAPS + attention_req) = 0x20;
    /* get the address of the interrupt function */
    getfunction(hold, iface_interrupt);
    /* point the interrupt to it */
    getw(FAREATB + attention_req) = hold;
    /* set leading edge interrupt */
    getw(FLEADTB + attention_req) = 0xffff;
    /* clear trailing edge interrupt */
    getw(FTRAILTB + attention_req) = 0;

```

```

/* clear both edge interrupt */
getw(FBOTHTB + attention_req) = 0;

}/* end of iface_init */

/* ----- */
void iface_clear() /* This routine called when CLEAR is pushed */
{
} /* end of iface_clear */

/* ----- */

void iface_main() /* Main interface handler */
{
/* Maintain internal feedhold if FAST STOP is active */
if (feed_hold_flag == 1) mcl_feedhold_on ();

if (getw(SWPTYPE) == 0) /* If routine is called by MAIN */
{
/* Acknowledge completion of the FAST STOP if necessary */
if ((int_cmd == 0x00fd) && (fast_stop_state == 2))
{
int_cmd = 0;
acknowledge ();
}

/* Acknowledge clearing of the FAST STOP if necessary */
if ((int_cmd == 0x00fc) && (fast_stop_state == 5))
{
mcl_feedhold_off ();
feed_hold_flag = 0;
int_cmd = 0;
acknowledge ();
}

/* Data Transfer is begun if the interrupt command = 0xfe */
if (int_cmd == 0x00fe)
{
rpc_ptr = 0; /* Initialize table pointer */

/* Read command word */
error_code = read_word(RPC_COMMAND_ADD);

/* Read input data type */
error_code = read_word(INP_DATA_TYPE_ADD);

switch(input_data_type)
{
case rpc_short: /* read one word */
error_code = read_word(RPC_DATA_ADD);

```

```

    break;
case rpc_long: /* read two words */
    error_code = read_word(RPC_DATA_ADD);
    error_code = read_word(RPC_DATA_ADD + 1);
    break;
case rpc_short_ary:
    break;
case rpc_long_ary:
    break;
case rpc_void: /* read no words */
    break;
default:
    break;
}

/* Read output data type */
error_code = read_word(OUT_DATA_TYPE_ADD);

```

```

/*****
/* Place SUPERVISION ROUTINES in switch after this comment */
/* NOTE: The fast stop routine and changing the programmed */
/* feedrate override are handled separately! */
*****/

```

```

switch(rpc_command)
{
case ext_feedhold: /* External feed hold */
    set (FDHOLDRQ);
    break;
case speed_change: /* Change spindle speed */
    getw(CURS) = getw(RPC_DATA_ADD);
    break;
case get_speed: /* Get spindle speed */
    getw(RPC_DATA_ADD) = getw(SPDRPM);
    break;
case get_feed: /* Get current FWORD */
    /* FWORD = (inches/min)*1000 */
    getd(RPC_DATA_ADD) = getd(CURF);
    break;
case get_feed_ovrd: /* Get manual feed override */
    getw(RPC_DATA_ADD) = getw(MFOP);
    break;
default:
    break;
}

```

```

/*****
/* Do not modify code after this comment */
*****/

```

```

/* Write back data, if necessary */
switch(output_data_type)
{
  case rpc_short: /* write one word */
    error_code = write_word(RPC_DATA_ADD);
    break;
  case rpc_long: /* write two words */
    error_code = write_word(RPC_DATA_ADD);
    error_code = write_word(RPC_DATA_ADD + 1);
    break;
  case rpc_short_ary:
    break;
  case rpc_long_ary:
    break;
  case rpc_void: /* write no words */
    break;
  default:
    break;
} /* end of switch(output_data_type) */
int_cmd = 0; /* Set int_cmd to FALSE */
} /* end of if(int_cmd == 0x00fe) */
return; /* Exit if run from MAIN */

} /* end of if(getw(SWPTYPE) == 0) */

} /* end of iface_main */

/* ----- */

void iface_interrupt()
/* This routine is run on the interrupt by the FIB card */
{
  /* Read in data from interrupt */
  input(input_port_address, input_data_reg);
  int_cmd = input_data_address & 0x00ff;

  /* Execute FAST STOP if requested */
  if (int_cmd == 0x00fd)
  {
    /* The FAST STOP is managed in stop.c (OEM_SYNC) */
    getw(NORMPFL) = -1; /* Step change in cmd generation */
    mcl_feedhold_on (); /* Stop interpolation by feedhold */
    getw(X1INHDPPE) = -1; /* Zero volts to X axis servo */
    getw(X2INHDPPE) = -1; /* Zero volts to Y axis servo */
    pfo_value_set(1); /* One percent feedrate override */
    feed_hold_flag = 1;
    fast_stop_flag = 1;
    fast_stop_state = 0;
  }

  /* Clear FAST STOP feedhold if requested */
  if ((fast_stop_flag == 1) && (int_cmd == 0x00fc))

```

```

{
/* The FAST STOP is cleared in stop.c (OEM_SYNC) */
fast_stop_state = 3;
}

/* Set programmed feedrate override (200 percent max).      */
/* The feedrate override is maintained in stop.c (OEM_SYNC) */
/* NOTE: The feedrate change is not acknowledged.          */
if ((int_cmd > 0) && (int_cmd < 0x00c9))
{
    pfo_percent = int_cmd;
    pfo_value_set (pfo_percent);
    int_cmd = 0;
}

/* Execute the interface communication protocol */
iface_main();
clear(SWPTYPE);

} /* end of iface_interrupt */

/* ----- */

int read_word(address)
    int address;
/* Read 2 bytes from the PC and store into one word at the */
/* specified address. Will eventually return error code.   */
{
    iface_set_bit(TR_BIT_MASK);          /* Set the TR bit */

    for(;;) { /* wait for ack bit to be set */
        input(input_port_address,input_data_reg);
        if((input_data_address & ACK_BIT_MASK) != 0) break;
    }

    /* read first byte */
    getw(address) = input_data_address & 0x00ff;

    iface_clear_bit(NOT_TR_BIT_MASK);    /* clear the TR bit */

    for(;;) { /* wait for ack bit to be cleared */
        input(input_port_address,input_data_reg);
        if((input_data_address & ACK_BIT_MASK) == 0) break;
    }

    iface_set_bit(TR_BIT_MASK);          /* Set the TR bit */

    for(;;) { /* wait for ack bit to be set */
        input(input_port_address,input_data_reg);
        if((input_data_address & ACK_BIT_MASK) != 0) break;
    }
}

```

```

/* read second byte */
getw(address) |= (input_data_address & 0x00ff) * 256;

iface_clear_bit(NOT_TR_BIT_MASK);      /* clear the TR bit */

for(;;) { /* wait for ack bit to be cleared */
    input(input_port_address,input_data_reg);
    if((input_data_address & ACK_BIT_MASK) == 0) break;
}
return (0);

} /* end of read_word */

/* ----- */

int write_word(address)
int address;
/* Write 2 bytes to the PC that come from the specified */
/* memory address. Will eventually return an error code. */
{
/* set data lines with lower byte from address */
output_data_address = ((getw(address)) & 0x00ff);
unpack(output_data_address,SCT + output_port_address,16);
outport(output_port_address,output_data_address);

iface_set_bit(TR_BIT_MASK);          /* Set the TR bit */

for(;;) { /* wait for ack bit to be set */
    input(input_port_address,input_data_reg);
    if((input_data_address & ACK_BIT_MASK) != 0) break;
}

iface_clear_bit(NOT_TR_BIT_MASK);    /* clear the TR bit */

for(;;) { /* wait for ack bit to be cleared */
    input(input_port_address,input_data_reg);
    if((input_data_address & ACK_BIT_MASK) == 0) break;
}

/* set data lines with high byte from address */
output_data_address = ((getw(address)) / 256);
unpack(output_data_address,SCT + output_port_address,16);
outport(output_port_address,output_data_address);

iface_set_bit(TR_BIT_MASK);          /* Set the TR bit */

for(;;) { /* wait for ack bit to be set */
    input(input_port_address,input_data_reg);
    if((input_data_address & ACK_BIT_MASK) != 0) break;
}

iface_clear_bit(NOT_TR_BIT_MASK);    /* clear the TR bit */

```

```

for(;;) { /* wait for ack bit to be cleared */
    input(input_port_address,input_data_reg);
    if((input_data_address & ACK_BIT_MASK) == 0) break;
}
return (0);

} /* end of write_word */

/* ----- */

void iface_set_bit(mask)
int mask;
/* Set a bit on the output port using the supplied mask */
{
    output_data_address = pack(SCT + output_port_address,16);
    output_data_address |= mask;
    unpack(output_data_address,SCT + output_port_address,16);
    outputport(output_port_address,output_data_address);
} /* end of iface_set_bit */

/* ----- */

void iface_clear_bit(mask)
int mask;
/* Clear a bit on the output port using the supplied mask */
{
    output_data_address = pack(SCT + output_port_address,16);
    output_data_address &= mask;
    unpack(output_data_address,SCT + output_port_address,16);
    outputport(output_port_address,output_data_address);
} /* end of iface_clear_bit */

/* ----- */

void acknowledge ()
/* This routine can be used to acknowledge an interrupt */
/* without initiating a data transfer through iface_main. */
{
    iface_set_bit(TR_BIT_MASK); /* Set the TR bit */

    for(;;) /* Wait for ack bit to be set */
    {
        input(input_port_address,input_data_reg);
        if((input_data_address & ACK_BIT_MASK) != 0) break;
    }
    iface_clear_bit(NOT_TR_BIT_MASK); /* Clear the TR bit */
} /* end of acknowledge */

/* ----- */

```



Fast Stopping Program

```

/*****
/*  Title:   STOP.C (Fast stopping routine for machine   */
/*          supervision system, Univ. of Florida)   */
/*  Date:    05-31-91                                   */
/*  Author:  R.L. Wells                                   */
/*  Version: 1.0                                         */
*****/

#include <mcl.h>
#include <globals.h>
#define STOP
#include <stdlib.h>
#include "oem_dec.h"

#define XERR                0x1825
#define axis_ep_value(A)   getd(&((long *)XERR)[A])
#define XCOMM               0x17dd
#define axis_cmd_posn(A)   getd(&((long *)XCOMM)[A])
#define XFBK                0x1801
#define axis_fbk_posn(A)   getd(&((long *)XFBK)[A])
#define XREF                0x17e9
#define axis_ref_posn(A)   getd(&((long *)XREF)[A])

#pragma macro dacout([index,=],[value, _])
$asm    lda    index                ; /* NOTE:          */
$asm    add    A                    ; /* TABS bewteen */
$asm    add    =X'1A56'             ; /* each of the  */
$asm    lde    *A                  ; /* elements in  */
$asm    lda    value               ; /* these lines. */
$asm    ioa    *E                  ; /* BEWARE!     */
#pragma endmacro

long last_x_fbk, last_y_fbk, test_x, test_y;
int  move, x_in_position, y_in_position;

/* ----- */

void stop_init() /* This routine is called once at power up */
{
fast_stop_state = 0; feed_hold_flag = 0; test_x = 0; test_y = 0;
last_x_fbk = 0; last_y_fbk = 0; pfo_percent = 0; move = 0;
x_in_position = 0; y_in_position = 0;

} /* end of stop_init */

/* ----- */

```

```

void stop()
{
if (fast_stop_flag != 1) /* If fast stop is not active */
{
/* Maintain the current programmed feedrate override */
pfo_value_set (pfo_percent);
}
if (fast_stop_flag == 1) /* If fast stop requested */
{
/* Keep the programmed feedrate override at 1 percent */
pfo_value_set (1);

/* Use the switch function as a state controller to */
/* synchronize the axis monitoring with SYNC passes. */
switch (fast_stop_state)
{
case 0:
x_in_position = 0;
y_in_position = 0;
fast_stop_state = 1;
/* No break for this case, code continues to case 1 */

case 1: /* Be sure the X and Y axes stop moving */
test_x = labs(axis_fbk_posn(0) - last_x_fbk);
test_y = labs(axis_fbk_posn(1) - last_y_fbk);
if ((test_x > 2) && (test_y > 2)) /* Tenths */
fast_stop_state = 1;
else
fast_stop_state = 2;
last_x_fbk = axis_fbk_posn(0);
last_y_fbk = axis_fbk_posn(1);
break;

case 2:
/* This fast_stop_state waits for the clear_fast_stop */
break;

case 3:
/* Fast_stop_state is set equal to 3 by the */
/* ClearFastStop int_cmd in iface.c (OEM_MAIN) */

getw(X1INHDPPE) = 1; /* Enable X axis DAC for output */
getw(X2INHDPPE) = 1; /* Enable Y axis DAC for output */
test_x = axis_ref_posn(0) - axis_fbk_posn(0);
test_y = axis_ref_posn(1) - axis_fbk_posn(1);

if (test_x > 5) /* Tenths */
{
move = 0x0180; /* 384 DAC units */
dacout(0, move);
}
if (test_x < -5) /* Tenths */

```

```

    {
        move = 0x8180; /* -384 DAC units */
        dacout(0, move);
    }
    if (test_y > 5) /* Tenths */
    {
        move = 0x0180; /* 384 DAC units */
        dacout(1, move);
    }
    if (test_y < -5) /* Tenths */
    {
        move = 0x8180; /* -384 DAC units */
        dacout(1, move);
    }
    if (labs(test_x) <= 10)
    {
        getw(X1INHDPE) = -1;
        x_in_position = 1;
    }
    if (labs(test_y) <= 10)
    {
        getw(X2INHDPE) = -1;
        y_in_position = 1;
    }
    if ((x_in_position == 1) && (y_in_position == 1))
    {
        getw(NORMPFL) = 0;
        /* XCOMM table must be updated to XREF table */
        axis_cmd_posn(0) = axis_ref_posn(0);
        axis_cmd_posn(1) = axis_ref_posn(1);
        fast_stop_state = 4;
    }
    break;

    case 4: /* Reconnect the X and Y axis servos */
        getw(X1INHDPE) = 0;
        getw(X2INHDPE) = 0;
        fast_stop_state = 5;
        fast_stop_flag = 0;
        break;

    /* Completion of the fast stop is acknowledged in */
    /* iface.c (OEM_MAIN) when fast_stop_state = 5 */
    default:
        break;

    } /* end of switch(fast_stop_state) */

} /* end of if(fast_stop_flag) */

} /* end of stop */
/* ----- */

```

APPENDIX B  
LISTING OF THE SPINDLE TORQUE OVERLOAD PROGRAM

```

/*****
/* TORQUE.C -- By: R.L. Wells and J. Frost    Date: 05-28-91  */
/* This program monitors the spindle encoder of the Omnimil  */
/* and declares a torque overload if the actual speed differs */
/* from the commanded speed by more than a specified amount. */
/* Include torque.c, pciface.c and dt2818.c in the make file. */
/*****

#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <graph.h>
#include <math.h>

#define LPT1M    0x037a
#define PICTRL  0x20
#define PICMSK  0x21
#define TMODE   0x43
#define TODATA  0x40
#define MAX     32000

#define MK_FP(seg,ofs)  ((void far *) \
                        (((unsigned long)(seg) << 16) | (unsigned)(ofs)))

/* Function prototypes. */

void startclk();
void enaclk();
void enalpt();
void disaclk();
void disalpt();
void far *getiv();
void setiv();
void far interrupt clk();
void far interrupt lptlack();

unsigned int outctr = 0, outrem = 0, ctr = 0;

/* ----- */

```

```

void main()
{
void far  *dosclk ;
void far  *doslpt ;
double    count = 0.0, speed = 0.0, trigspeed = 0.0;
float     speedout = 0.0, pth = 0.0, thresh = 0.50;
int       ncspeed = 0, testspeed = 0, cnt, pfo = 100;
int       startflag = 0;
char      a;

/* Initialize the digital I/O port (MCP interface). */
Dt2817Init ();

start:

startflag = 0;
_clearscreen(_GCLEARSCREEN); _settextposition (1,1);
printf ("*** SPINDLE TORQUE OVERLOAD DETECTION ***\n\n");

/* Wait for a spindle speed to be programmed to the MCP. */
while (1)
{
  GetSpindleSpeed (&ncspeed);
  _settextposition (3,1);
  printf ("Commanded CNC spindle speed = %4d (RPM)", ncspeed);
  if( kbhit() )
  {
    _settextposition (11,1);
    printf ("\n\nSystem terminated by user.\n");
    exit (1);
  }
  if (ncspeed >10) break;
}

/* Enter the spindle speed threshold */
_settextposition (5,1);
printf ("Enter the trigger threshold: (%% of CNC speed) ");
scanf ("%f", &pth);
thresh = pth / 100;
trigspeed = (double)(ncspeed * thresh);

/* Set up the tachometer by redirecting the */
/* SYSTEM CLOCK and LPT1 interrupt vectors */
dosclk = getiv(0x0008);
setiv( 0x0008, clk );
doslpt = getiv(0x000f);
setiv( 0x000f, lptlack );
enaclk();
enalpt();

SetFeedOvrd (pfo);

```

```

/* Wait for the spindle to get up to speed. */
ctr = 0; count = 0.0; cnt = 0; speed = 0.0;
while(1)
{
    count = (double)outctr * 65536.0 + (65536.0-(double)outrem);
    if (count <= 0.0) count = 1.0;
    speed = 1.1931817e6 / count * 60.0;
    if ((speed > trigspeed) && (cnt > 1000)) break;
    cnt += 1; if (cnt > MAX) cnt = 0;
}

/* ***** Monitor Spindle Speed ***** */

ctr = 0; count = 0.0; cnt = 0; speed = 0.0;
while(1)
{
    /* Calculate the trigger speed. */
    trigspeed = (double)(ncspeed * thresh);
    _settextposition (7, 1);
    printf ("Minimum allowed speed = % 4.0lf (RPM) ", trigspeed);

    /* Read the tachometer. */
    count = (double)outctr * 65536.0 + (65536.0-(double)outrem);
    if (count <= 0.0) count = 1.0;
    speed = 1.1931817e6 / count * 60.0;
    _settextposition (9,1);
    printf("Actual Spindle Speed = % 4.0lf (RPM)          ", speed );

    /* Write speed on D/A channel 0 for data sampling. */
    speedout = speed / 500.0;
    write_da(0, &speedout);

    /* Update the commanded spindle speed. */
    GetSpindleSpeed (&ncspeed);
    _settextposition (3, 1);
    printf ("Commanded CNC spindle speed = %4d (RPM)", ncspeed);

    /* Generate a fast stop if speed drops below the limit. */
    if ((speed <= trigspeed) && (cnt > 2))
    {
        /* Protect against false triggers. */
        GetSpindleSpeed (&testspeed);
        if (testspeed <= trigspeed)
        {
            startflag = 1;
            goto cont;
        }
        trigspeed = (double)(testspeed * thresh);
        count = (double)outctr * 65536.0 + (65536.0-(double)outrem);
        if (count <= 0.0) count = 1.0;
        speed = 1.1931817e6 / count * 60.0;
        if (testspeed != ncspeed) goto skip;
    }
}

```

```

if (speed >= trigspeed) goto skip;
FastStop ();
_settextposition (11,1);
printf ("SPINDLE SPEED BELOW ALLOWED LIMIT!\n");
printf ("\nFast stop command issued to the MCP.\n");
printf ("\nPress ANY KEY to clear the fast stop: ");
while (!kbhit())
{
    count = (double)outctr * 65536.0
            + (65536.0-(double)outrem);
    if (count <= 0.0) count = 1.0;
    speed = 1.1931817e6 / count * 60.0;
    speedout = speed / 500.0;
    write_da(0, &speedout);
}
getch(); fflush(stdin);
ClearFastStop();
printf ("\n");
startflag = 1;
goto cont;
}

skip:

if( kbhit() )
{
    _settextposition (11,1);
    printf ("System terminated by user.\n");
    getch(); fflush(stdin);
    break;
}
cnt += 1; if (cnt > MAX) cnt = 0;
}

cont:

/* Restore the SYSTEM CLOCK and LPT1 interrupts. */
disalpt();
disaclk();
setiv( 0x0008, dosclk );
startclk();
enaclk();
setiv( 0x000f, doslpt );
if (startflag == 1) goto start;

SetFeedOvrd (pfo);
exit (1);

} /* end of main */

/* ----- */

```

```

void enaclk()
{
_asm in al, PICMSK ; /* unmask PIC timer bit */
_asm and al, 0feh ;
_asm out PICMSK, al ;
} /* end of enaclk */

/* ----- */

void enalpt()
{
_asm in al, PICMSK ; /* unmask PIC // port bit */
_asm and al, 07fh ;
_asm out PICMSK, al ;
_asm mov dx,LPT1M ; /* enable // interupt */
_asm in al,dx ;
_asm or al, 010h ;
_asm out dx, al ;
} /* end of enalpt */

/* ----- */

void disaclk()
{
_asm in al, PICMSK ; /* mask PIC serial bit */
_asm or al, 01h ;
_asm out PICMSK, al ;
} /* end of disaclk */

/* ----- */

void disalpt()
{
_asm in al, PICMSK ; /* mask PIC // port bit */
_asm or al, 080h ;
_asm out PICMSK, al ;
_asm mov dx,LPT1M ; /* disable // interupt */
_asm in al,dx ;
_asm and al, 0efh ;
_asm out dx, al ;
} /* end of disalpt */

/* ----- */

void far *getiv( unsigned inum )
{
unsigned far *source;
source = MK_FP( 0, inum<<2 );
return( MK_FP( *(source+1), *source ) );
} /* end of *getiv */

/* ----- */

```



```

void setiv( unsigned inum, void far *fptr )
{
  unsigned far *dest;
  _asm cli ;
  _dest = MK_FP( 0, inum<<2 ) ;
  *dest = FP_OFF(fptr) ;
  dest++ ;
  *dest = FP_SEG(fptr) ;
  _asm sti ;
} /* end of setiv */

/* ----- */

void far interrupt clk()
{
  _asm inc ctr ; /* increment wrap counter */
  _asm mov al,030h ; /* pmg. for write 2 bytes */
  _asm out TMODE,al ;
  _asm mov al,0 ;
  _asm out TODATA,al ; /* load for full count */
  _asm out TODATA,al ;
  _asm mov al,20h ; /* end of interrupt */
  _asm out PICTRL,al ;
} /* end of clk */

/* ----- */

void startclk() /* reprogram timer-0 for mode 3 */
{
  _asm mov al,036h ; /* pmg. for write 2 bytes */
  _asm out TMODE,al ;
  _asm mov al,0 ;
  _asm out TODATA,al ; /* load for full count */
  _asm out TODATA,al ;
} /* end of startclk */

/* ----- */

void far interrupt lptlack()
{
  _asm mov al,0 ; /* latch counter */
  _asm out TMODE,al ;
  _asm mov al,030h ; /* pmg. for read 2 bytes */
  _asm out TMODE,al ;
  _asm in al,TODATA ; /* read low byte */
  _asm mov ah,al ;
  _asm in al,TODATA ; /* read hi byte */
  _asm xchg al,ah ;
  _asm mov outrem,ax ; /* store remanider */
  _asm mov ax,ctr ;
  _asm mov outctr,ax ; /* store # wraps */
  _asm mov ax,0 ;
}

```

```
_asm mov ctr,ax ; /* zero wrap counter */
_asm mov al,030h ; /* pmg. for write 2 bytes */
_asm out TMODE,al ;
_asm mov al,0 ;
_asm out TODATA,al ; /* load for full count */
_asm out TODATA,al ;
_asm mov dx,LPT1M ; /* enable // interupt */
_asm mov al, 010h ;
_asm out dx, al ;
_asm mov al,20h ; /* end of interrupt */
_asm out PICTRL,al ;
} /* end of lptlack */

/* ----- */
```

APPENDIX C  
LISTING OF THE MACHINE EVALUATION PROGRAM

```

REM *****
REM * PCDATA.BAS      : ver 1.0                      *
REM * WRITTEN BY     : R.L. Wells, with thanks to J. Frost *
REM * LAST MODIFIED : 06-13-91                      *
REM * Use BCDATA.BAT to compile and LINKDATA.BAT to link. *
REM *****

```

```

' $DYNAMIC
' $INCLUDE: 'GWDECL.INC'

```

```
DEFINT A-Z
```

```

DECLARE SUB STOP.AND.CLEAR ()
DECLARE SUB CHECK.ERROR ()
DECLARE SUB SET.DMA.CONTROLLER (DMAMODE%, NC%)
DECLARE SUB SET.DMA.DT2818 (TICKS%, START.CHAN%, EMD.CHAN%)
DECLARE SUB WRITE.DA (DACHAN%, VOLTS!)
DECLARE SUB READ.AD (ADCHAN%, VAD!)

```

```

DECLARE SUB FFT CDECL (BYVAL O1, BYVAL S1, BYVAL O2, BYVAL S2,
BYVAL O3, BYVAL S3, BYVAL O4, BYVAL S4, BYVAL NN%)

```

```

DECLARE SUB MENU (A%)
DECLARE SUB PLOT1 (X!(), Y1!(), N%)
DECLARE SUB PLOT2 (X1!(), Y1!(), N1%, X2!(), Y2!(), N2%)
DECLARE SUB CURVE (X!(), Y!(), N%)
DECLARE SUB SMOOTH (M%, Z%, d!(), N%)
DECLARE SUB WRITE.BINARY (Filename$, DAT!(), TOTAL&, TIMESTEP!)
DECLARE SUB GETKEY (A$)

```

```

REM Global values for the DMA chip. DMACHAN, DMACOM, BASEREG,
REM COUNTREG and PAGEREG are redefined in the Setup Module.

```

```

COMMON SHARED DMACHAN, DMACOM, BASEREG, COUNTREG, PAGEREG
CONST DMAMASK = &H5, DMAPAGE = &H4, SEG.VAL = &H4000
CONST MASKREG = &HA, MODEREG = &HB, BYTEREG = &HC
CONST DMABASEL = &H0, DMABASEH = &H0

```

```

REM Global values for the DT-2818 board. BASE.ADR, COM.REG,
REM STAT.REG and DATA.REG are redefined in the Setup Module.

```

```

COMMON SHARED BASE.ADR, COM.REG, STAT.REG, DATA.REG
CONST COM.WAIT = &H4, WRITE.WAIT = &H2, READ.WAIT = &H5
CONST CSTOP = &HF, CCLEAR = &H1, CERROR = &H2, CCLOCK = &H3
CONST CRAD = &HE, CSAD = &HD, CDAOUT = &H8, CADIN = &HC
CONST FACTOR = 4096, RANGE = 20, OFFSET = 10
CONST RES! = RANGE / FACTOR

REM Default values: BASE.ADR = &H2EC, DMACHAN = 1, NMAX = 4096

BASE.ADR = &H2EC: COM.REG = BASE.ADR + 1
STAT.REG = BASE.ADR + 1: DATA.REG = BASE.ADR
DMACHAN = &H1: DMACOM = &H45: BASEREG = &H2
COUNTREG = &H3: PAGEREG = &H83: NMAX = 4096

REM Detect screen type and display the main menu.

START:
TEST! = FRE(""): VIEW PRINT 1 TO 25
OUT &H70, &H14
IF (INP(&H71) AND &H20) = 0 THEN SC$ = "E" ELSE SC$ = "C"

CALL MENU(CX)
SELECT CASE CX

CASE 1

REDIM SCL!(3), TITLE$(3)
FOR I = 0 TO 3: SCL!(I) = 1: NEXT
K = 1: I = 0: DV = 0: VOLTS! = 0: BASEFREQ = 25000
BASECOUNT = 25000: NCHAN = 1: Z0 = 0: Z1 = 0: Z2 = 0
Z3 = 0: T! = 0: NUM$ = "": SF = 0: ND2 = NMAX / 2: CHATMOD = 0

D4: CLS : VIEW PRINT 1 TO 25: LIM = 1: TW! = 0
TITLE$(0) = "CHANNEL 0": TITLE$(1) = "CHANNEL 1"
TITLE$(2) = "CHANNEL 2": TITLE$(3) = "CHANNEL 3"
PRINT "
PRINT "          DATA ACQUISITION MODULE          "
PRINT "
PRINT
PRINT "Four channels are available starting with CHANNEL 0."
PRINT
PRINT "How many CHANNELS do you want to read? (; NCHAN; ") ";
INPUT "", A$: IF A$ = "" THEN NCHAN = NCHAN ELSE NCHAN = VAL(A$)
IF NCHAN < 1 OR NCHAN > 4 THEN GOTO D9
PRINT
FOR I = 0 TO NCHAN - 1
  PRINT "Enter the SCALE for "; TITLE$(I); ": ";
  PRINT "(; SCL!(I); "units/volt ) "; : INPUT "", A$
  IF A$ = "" THEN SCL!(I) = SCL!(I) ELSE SCL!(I) = VAL(A$)
  IF SCL!(I) = 0 THEN GOTO D9
NEXT

```

```

D4X:
IF SF = 0 OR SF > BASEFREQ / NCHAN THEN
  SF = CINT(BASEFREQ / NCHAN)
END IF
IF CNT& = 0 OR CNT& > INT(BASECOUNT / NCHAN + .5) THEN
  CNT& = INT(BASECOUNT / NCHAN + .5)
END IF
PRINT
PRINT "Enter the SAMPLING RATE: ("; SF; "Hz ) ";
INPUT "", A$: IF A$ = "" THEN SF = SF ELSE SF = VAL(A$)
IF SF < 100 OR SF > CINT(BASEFREQ / NCHAN) THEN
  IF CHATMOD = 1 THEN GOTO C9 ELSE GOTO D9
END IF
TICKS = 800000 / SF: DT! = 1 / SF
PRINT
PRINT "Enter the NUMBER OF SAMPLES per channel: ("; CNT&; ") ";
INPUT "", A$: IF A$ = "" THEN CNT& = CNT& ELSE CNT& = VAL(A$)
IF CNT& < 1000 OR CNT& > INT(BASECOUNT / NCHAN + .5) THEN
  IF CHATMOD = 1 THEN GOTO C9 ELSE GOTO D9
END IF
NC = CNT& * NCHAN: NCM1 = NC - 1
PRINT
PRINT "The OBSERVATION TIME will be";
PRINT CLNG(NC * DT! / NCHAN * 10 ^ 3) / 10 ^ 3; "sec.";
PRINT " Continue? [*Y/N] "
CALL GETKEY(A$): IF A$ = "N" THEN GOTO D4X

REDIM TIME!(CNT&), DO!(CNT&)
IF NCHAN = 2 THEN REDIM D1!(CNT&)
IF NCHAN = 3 THEN REDIM D1!(CNT&), D2!(CNT&)
IF NCHAN = 4 THEN REDIM D1!(CNT&), D2!(CNT&), D3!(CNT&)

REM Ask if an external trigger is used.
EXT.TRIG = 0
PRINT : PRINT "Wait for an EXTERNAL TRIGGER? [Y/*N] "
CALL GETKEY(A$): IF A$ = "Y" THEN EXT.TRIG = &H80: GOTO D6

D5: REM The Chatter Analysis Module comes here.
PRINT
PRINT "Press RETURN to begin or Q to quit: ": CALL GETKEY(A$)
IF A$ = "Q" THEN IF CHATMOD THEN GOTO C9 ELSE GOTO D9

REM Disable the system clock.
D6: OUT &H43, &H30: OUT &H40, &H0: OUT &H40, &H0

REM Stop and clear the DT-2818.
CALL STOP.AND.CLEAR

REM Set-up DMA controller chip.
CALL SET.DMA.CONTROLLER(DMACOM, NC)

REM Set up DT-2818 for DMA operation.

```

```

CALL SET.DMA.DT2818(TICKS, 0, NCHAN - 1)

IF EXT.TRIG = 0 THEN
  PRINT : PRINT "Working ... ";
ELSE
  PRINT : PRINT "Waiting for trigger ... ";
END IF

REM Write READ A/D WITH DMA command.
WAIT STAT.REG, WRITE.WAIT, WRITE.WAIT
WAIT STAT.REG, COM.WAIT
OUT COM.REG, CRAD + &H10 + EXT.TRIG

REM Check for ERROR.
CALL CHECK.ERROR

REM Enable the system clock.
OUT &H43, &H36: OUT &H40, &H0: OUT &H40, &H0

REM Extract A/D readings from memory & scale.
T! = 0: Z0 = 1: Z1 = 1: Z2 = 1: Z3 = 1: PRINT "Processing ..."
DEF SEG = SEG.VAL
  FOR K = 0 TO NCM1
    DV = PEEK(K * 2) + PEEK(K * 2 + 1) * 256
    VOLTS! = RES! * DV - OFFSET
    IF K MOD NCHAN = 0 THEN
      TIME!(Z0) = T!: D0!(Z0) = VOLTS! * SCL!(0)
      Z0 = Z0 + 1
    END IF
    IF K MOD NCHAN = 1 THEN
      TIME!(Z1) = T!: D1!(Z1) = VOLTS! * SCL!(1)
      Z1 = Z1 + 1
    END IF
    IF K MOD NCHAN = 2 THEN
      TIME!(Z2) = T!: D2!(Z2) = VOLTS! * SCL!(2)
      Z2 = Z2 + 1
    END IF
    IF K MOD NCHAN = 3 THEN
      TIME!(Z3) = T!: D3!(Z3) = VOLTS! * SCL!(3)
      Z3 = Z3 + 1
    END IF
    IF (K MOD NCHAN) = NCHAN - 1 THEN T! = T! + DT!
  NEXT
DEF SEG

REM If data is for Chatter Analysis Module then go back there.
IF CHATMOD THEN GOTO C5

REM Plot the TIME data only.
D8: GOSUB PX1

REM Calculate FFT's and plot TIME data and SPECTRUM.

```



```

INPUT "", A$: IF A$ = "" THEN SF = SF ELSE SF = VAL(A$)
IF SF <= 0 OR SF > 12000 THEN GOTO T9
TICKS = 800000 / SF: DT! = 1 / SF: DF! = 1 / (NMAX * DT!)

REM Construct Frequency range.
FOR I = 1 TO ND2: F!(I) = I * DF!: NEXT

PRINT
FOR I = 0 TO 1
  PRINT "Enter the SCALE for "; TITLE$(I); ": ";
  PRINT "("; SCL!(I); "units/volt ) "; : INPUT "", A$
  IF A$ = "" THEN SCL!(I) = SCL!(I) ELSE SCL!(I) = VAL(A$)
  IF SCL!(I) = 0 THEN GOTO T9
NEXT

PRINT : PRINT "Enter the TRIGGER THRESHOLD: ("; TH!; "volts ) ";
INPUT "", A$: IF A$ = "" THEN TH! = TH! ELSE TH! = VAL(A$)

PRINT : PRINT "Enter the NUMBER OF AVERAGES: ("; NAVG; ") ";
INPUT "", A$: IF A$ = "" THEN NAVG = NAVG ELSE NAVG = VAL(A$)
IF NAVG = 0 OR NAVG > 100 THEN GOTO T9

PRINT : PRINT "Press RETURN to begin or Q to quit: ";
CALL GETKEY(A$): IF A$ = "Q" THEN PRINT : GOTO T9
PRINT

REM Disable the Time-of-Day clock.
OUT &H43, &H30: OUT &H40, &H0: OUT &H40, &H0

REM Begin taking the Transfer Functions.
FOR I = 1 TO NAVG

REM Refer trigger threshold to DC offset of hammer charge amp.
T3: CALL STOP.AND.CLEAR: CALL READ.AD(0, TEST!)
  LIMIT = FACTOR / 2 + TH! / RES! + TEST! / RES!
  CALL STOP.AND.CLEAR

REM Set-up DT-2818 for continuous DMA operation
  CALL SET.DMA.DT2818(TICKS, 0, 1)

REM Set-up DMA controller for memory write, autoinitialize.
  CALL SET.DMA.CONTROLLER(DMACOM + &H10, NC)

REM Write READ A/D WITH CONTINUOUS DMA command.
  WAIT STAT.REG, COM.WAIT
  OUT COM.REG, CRAD + &H30

REM Monitor DMA memory and look for the trigger on Channel 0.
  SCREEN 0: CLS : LOCATE 12, 19
  PRINT "Waiting for Trigger #"; I; "(ESC to Exit) ... ";
  LAST = (DMABASEH * 256 + DMABASEL) \ 2
T5: IF INKEY$ = CHR$(27) THEN

```



```

    OUT &H43, &H36: OUT &H40, &H0: OUT &H40, &H0
    CALL STOP.AND.CLEAR: GOTO T1
END IF
OUT MASKREG, DMAMASK: REM Set DMA channel 1 mask bit.
OUT BYTEREG, DMACHAN: REM Clear byte flipflop.
CL& = INP(BASEREG): CH& = INP(BASEREG)
OUT MASKREG, DMACHAN: REM Clear DMA channel 1 mask bit.
CURRENT = (CH& * 256 + CL&) \ 2
IF CURRENT > 0 THEN CURRENT = CURRENT - CURRENT MOD 2

DEF SEG = SEG.VAL
  IF LAST < CURRENT THEN
    FOR K = LAST TO CURRENT STEP 2
      DV = PEEK(K * 2) + PEEK(K * 2 + 1) * 256
      IF DV > LIMIT THEN
        DEF SEG : GOTO T6: REM Trigger found!
      END IF
    NEXT
  ELSE
    FOR K = LAST TO NCM1 STEP 2
      DV = PEEK(K * 2) + PEEK(K * 2 + 1) * 256
      IF DV > LIMIT THEN
        DEF SEG : GOTO T6: REM Trigger found!
      END IF
    NEXT
    FOR K = 0 TO CURRENT STEP 2
      DV = PEEK(K * 2) + PEEK(K * 2 + 1) * 256
      IF DV > LIMIT THEN
        DEF SEG : GOTO T6: REM Trigger found!
      END IF
    NEXT
  END IF
DEF SEG
LAST = CURRENT: GOTO T5: REM Trigger not found.

REM Poll DMA controller to be sure that data is not overwritten.
REM Reject premature trigger.
T6: IF K < 4 OR CURRENT < 4 THEN GOTO T5
REM Pre-trigger by 3 data values to catch entire impulse.
IF K < 6 THEN K = NCM1 + K - 6 ELSE K = K - 6
IF K <= NCM1 - NMAX2M1 THEN
  ENDVAL = K + NMAX2M1
ELSE
  ENDVAL = NMAX2M1 - NCM1 + K
END IF
IF K <= NCM1 - NMAX2M1 THEN
  ADR = K
  DO WHILE ADR < ENDVAL
    OUT MASKREG, DMAMASK: OUT BYTEREG, DMACHAN
    CL& = INP(BASEREG): CH& = INP(BASEREG)
    OUT MASKREG, DMACHAN
    ADR = (CH& * 256 + CL&) \ 2
  
```

```

      LOOP
ELSE
  ADR = K
  DO WHILE ADR < NCM1
    OUT MASKREG, DMAMASK: OUT BYTEREG, DMACHAN
    CL& = INP(BASEREG): CH& = INP(BASEREG)
    OUT MASKREG, DMACHAN
    ADR = (CH& * 256 + CL&) \ 2
  LOOP
  ADR = 0
  DO WHILE ADR < ENDVAL
    OUT MASKREG, DMAMASK: OUT BYTEREG, DMACHAN
    CL& = INP(BASEREG): CH& = INP(BASEREG)
    OUT MASKREG, DMACHAN
    ADR = (CH& * 256 + CL&) \ 2
  LOOP
END IF

REM Stop the DMA process.
OUT COM.REG, CSTOP: BEEP

REM Check for DT-2818 error.
CALL CHECK.ERROR

REM Recover data from memory.
CLS : LOCATE 12, 26: PRINT "Processing (ESC to Hold) ..."
CNT = 0: Z0 = 1: Z1 = 1: OVLD = 0
DEF SEG = SEG.VAL
  REM Set up square window for hammer impulse on channel 0.
  ADR = K + FW * 2
  IF ADR > NCM1 THEN ADR = FW * 2 + K - NCM1
  OFS! = PEEK(ADR * 2) + PEEK(ADR * 2 + 1) * 256
  OFS! = (RES! * OFS! - OFFSET) * SCL!(0)
  IF K < ENDVAL THEN
    FOR J = K TO ENDVAL
      DV = PEEK(J * 2) + PEEK(J * 2 + 1) * 256
      IF (DV <= 2 OR DV >= 4094) AND OVLD = 0 THEN OVLD = 1
      VOLTS! = RES! * DV - OFFSET
      IF CNT MOD 2 = 0 THEN
        D0!(Z0) = VOLTS! * SCL!(0)
        IF Z0 > FW THEN D0!(Z0) = OFS!: REM Apply window.
        Z0 = Z0 + 1
      ELSE
        D1!(Z1) = VOLTS! * SCL!(1): Z1 = Z1 + 1
      END IF
      CNT = CNT + 1
      IF Z0 > NMAX AND Z1 > NMAX THEN EXIT FOR
      IF INKEY$ = CHR$(27) THEN DEF SEG : GOTO T3
    NEXT
  ELSE
    FOR J = K TO NCM1
      DV = PEEK(J * 2) + PEEK(J * 2 + 1) * 256

```

```

IF (DV <= 2 OR DV >= 4094) AND OVLD = 0 THEN OVLD = 1
VOLTS! = RES! * DV - OFFSET
IF CNT MOD 2 = 0 THEN
  DO!(Z0) = VOLTS! * SCL!(0)
  IF Z0 > FW THEN DO!(Z0) = OFS!: REM Apply window.
  Z0 = Z0 + 1
ELSE
  D1!(Z1) = VOLTS! * SCL!(1): Z1 = Z1 + 1
END IF
CNT = CNT + 1
IF INKEY$ = CHR$(27) THEN DEF SEG : GOTO T3
NEXT
FOR J = 0 TO ENDVAL
  DV = PEEK(J * 2) + PEEK(J * 2 + 1) * 256
  IF (DV <= 2 OR DV >= 4094) AND OVLD = 0 THEN OVLD = 1
  VOLTS! = RES! * DV - OFFSET
  IF CNT MOD 2 = 0 THEN
    DO!(Z0) = VOLTS! * SCL!(0)
    IF Z0 > FW THEN DO!(Z0) = OFS!: REM Apply window.
    Z0 = Z0 + 1
  ELSE
    D1!(Z1) = VOLTS! * SCL!(1): Z1 = Z1 + 1
  END IF
  CNT = CNT + 1
  IF Z0 > NMAX AND Z1 > NMAX THEN EXIT FOR
  IF INKEY$ = CHR$(27) THEN DEF SEG : GOTO T3
NEXT
END IF
DEF SEG

IF OVLD THEN
  OVLD = 0
  CLS : LOCATE 12, 24
  PRINT "OVERLOAD! Continue or Hold? [C/*H] "
  CALL GETKEY(A$): IF A$ <> "C" THEN GOTO T3
  CLS : LOCATE 12, 24
  PRINT "Calculating Transfer Function ..."
END IF

REM Subtract DC offset & apply exponential window.
SUM0! = 0: SUM1! = 0
FOR J = 1 TO NMAX
  SUM0! = SUM0! + DO!(J)
  SUM1! = SUM1! + D1!(J)
  IF INKEY$ = CHR$(27) THEN GOTO T3
NEXT
AVG0! = SUM0! / NMAX
AVG1! = SUM1! / NMAX
TAU! = .1 * DT! * NMAX
FOR J = 1 TO NMAX
  DO!(J) = (DO!(J) - AVG0!) * EXP(-J * DT! / TAU!)
  D1!(J) = (D1!(J) - AVG1!) * EXP(-J * DT! / TAU!)

```

```

      IF INKEY$ = CHR$(27) THEN GOTO T3
NEXT

REM Calculate FFT of the impact and the response.
CLS : LOCATE 12, 24: PRINT "Calculating Transfer Function ..."
CALL FFT(VARPTR(DO!(0)), VARSEG(DO!(0)), VARPTR(REO!(0)),
        VARSEG(REO!(0)), VARPTR(IMO!(0)), VARSEG(IMO!(0)),
        VARPTR(MAGO!(0)), VARSEG(MAGO!(0)), NMAX)
CALL FFT(VARPTR(D1!(0)), VARSEG(D1!(0)), VARPTR(RE1!(0)),
        VARSEG(RE1!(0)), VARPTR(IM1!(0)), VARSEG(IM1!(0)),
        VARPTR(MAG1!(0)), VARSEG(MAG1!(0)), NMAX)

REM Create average Transfer Functions Re[X1/X0] and Im[X1/X0].
REM Use Cross and Auto spectrum for T.F. computation.
J = I - 1: IF I = 1 THEN J = 1
FOR K = 1 TO ND2
  DENOM! = MAGO!(K) ^ 2: IF DENOM! = 0 THEN DENOM! = 1E-10
  REX!(K) = (REO!(K) * RE1!(K) + IMO!(K) * IM1!(K)) / DENOM!
  RE!(K) = (J * RE!(K) + REX!(K)) / I
  IMX!(K) = (REO!(K) * IM1!(K) - IMO!(K) * RE1!(K)) / DENOM!
  IM!(K) = (J * IM!(K) + IMX!(K)) / I
NEXT

REM Plot the HAMMER magnitude and the IMAG average TF.
XL1$ = "FREQ (Hz)": YL1$ = "MAG HAMMER"
XL2$ = XL1$: YL2$ = "IMAG AVG TF"
TEST! = MAGO!(1): MAGO!(1) = 0
LIM = 0: CALL PLOT2(F!(), MAGO!(), ND2, F!(), IM!(), ND2)
MAGO!(1) = TEST!

REM If HOLD then subtract the present TF from the average TF.
LOCATE 1, 1
PRINT "Continue, Hold, Stop or Average? [*C/H/S/A] "
CALL GETKEY(A$): VIEW: CLS
IF A$ = "H" THEN
  FOR K = 1 TO ND2
    RE!(K) = (I * RE!(K) - REX!(K)) / J
    IM!(K) = (I * IM!(K) - IMX!(K)) / J
  NEXT
  GOTO T3
END IF
IF A$ = "S" THEN
  SCREEN 0: I = NAVG
  OUT &H43, &H36: OUT &H40, &H0: OUT &H40, &H0
  CALL STOP.AND.CLEAR: GOTO T1
END IF
IF A$ = "A" THEN I = NAVG: EXIT FOR

REM Continue with the next TF.

NEXT I

```

```

REM Enable the Time-of-Day clock.
OUT &H43, &H36: OUT &H40, &H0: OUT &H40, &H0

REM Plot the REAL and IMAGINARY average Transfer Function.
XL1$ = "FREQ (Hz)": YL1$ = "REAL TF"
XL2$ = XL1$: YL2$ = "IMAG TF"
LIM = 1: CALL PLOT2(F!(), RE!(), ND2, F!(), IM!(), ND2)
SCREEN 0: CLS

REM Save the FREQUENCY, REAL and IMAGINARY TF to ASCII file.
PRINT : PRINT "Save the Transfer Function to disk? [Y/*N] "
CALL GETKEY(A$)
IF A$ = "Y" THEN
  PRINT
  INPUT "Enter the ASCII file name: ", FILE$
  IF FILE$ = "" THEN GOTO T1
  OPEN FILE$ FOR OUTPUT AS #1
  PRINT : PRINT "Saving "; FILE$; " ..."
  FOR I = 1 TO ND2
    PRINT #1, USING "#.#####^ ^ ^ "; F!(I); RE!(I); IM!(I)
  NEXT
  CLOSE #1
END IF

T9: ERASE DO!, D1!, F!, RE!, IM!, REX!, IMX!
  ERASE RE0!, IM0!, MAG0!, RE1!, IM1!, MAG1!
PRINT : PRINT "Continue the TRANSFER FUNCTION module? [*Y/N] "
CALL GETKEY(A$): IF A$ = "N" THEN GOTO START ELSE GOTO T1

CASE 3

REDIM SCL!(1), TITLE$(1)
K = 1: I = 0: DV = 0: VOLTS! = 0: Z0 = 0: T! = 0: TW! = 0
EXT.TRIG = 0: LIM = 1: SCL!(0) = 1: ACSPD! = 3600
NT = 4: THR = 15
SCL!(0) = 1: TITLE$(0) = "CHANNEL 0": NUM$ = ""
C1: CLS : VIEW PRINT 1 TO 25
MAXTEMP = NMAX: NMAX = 8192: ND2 = NMAX / 2: SF = 2 * NMAX
NCHAN = 1: CNT& = NMAX: NC = CNT& * NCHAN: NCM1 = NC - 1
TICKS = 800000 / SF: DT! = 1 / SF
PRINT "
PRINT "
PRINT "
          CHATTER ANALYSIS MODULE
          "
          "
REDIM TIME!(CNT&), DO!(CNT&), DX!(CNT&)
REDIM F!(ND2), SRT!(ND2), FRQ!(ND2), MAG!(ND2), TEMP!(ND2)
PRINT
PRINT "Enter the NUMBER OF TEETH on the TOOL: (; NT; ) ";
INPUT "", A$: IF A$ = "" THEN NT = NT ELSE NT = VAL(A$)
IF NT < 1 OR NT > 100 THEN GOTO C1
PRINT
PRINT "Enter the ACTUAL SPINDLE SPEED: (; ACSPD!; "RPM ) ";
INPUT "", A$

```

```

IF A$ = "" THEN ACSPD! = ACSPD! ELSE ACSPD! = VAL(A$)
IF ACSPD! < 1 OR ACSPD! > 40000 THEN GOTO C1

REM Construct Frequency range.
DF! = 1 / (NMAX * DT!)
FOR I = 1 TO ND2: F!(I) = I * DF!: NEXT

REM Determine search step for the spectrum.
FC! = 0: FT! = ACSPD! * NT / 60
FR! = ACSPD! / 60: MAX! = -1E+30
FD! = 3 * DF!
IF 2 * FD! > FR! THEN FD! = 2 * DF!
IF 2 * FD! > FR! THEN
  BEEP: PRINT
  PRINT "Spindle speed is TOO SLOW to resolve chatter!"
  GOTO C9
END IF

PRINT
PRINT "Enter the THRESHOLD for chatter detection : (";
PRINT THR; "% ) "; : INPUT "", A$
IF A$ = "" THEN THR = THR ELSE THR = VAL(A$)
IF THR <= 0 OR THR > 100 THEN GOTO C1
PRINT
PRINT "The MAXIMUM FREQUENCY will be"; SF / 2; "Hz, ";
PRINT "with a RESOLUTION of"; SF / NMAX; "Hz."
PRINT
PRINT "Check that the MICROPHONE is on CHANNEL 0."

REM Use the Data Acquisition Module to get the data in D0!().
CHATMOD = 1: GOTO D5

REM The Data Acquisition Module returns here.
C5: CHATMOD = 0

REM Get ready to subtract DC offset from the data.
SUM0! = 0
FOR I = 1 TO NMAX
  SUM0! = SUM0! + D0!(I)
NEXT
AVG0! = SUM0! / NMAX

REM Subtract DC offset and apply Hanning window.
FOR I = 1 TO NMAX
  DX!(I) = (D0!(I) - AVG0!) * .5 * (1 - COS(6.283185 * (I - 1) /
    (NMAX - 1)))
NEXT

REM Compute the FFT of the data.
PRINT
PRINT "Computing one"; NMAX; "point Fast Fourier Transform ..."
CALL FFT(VARPTR(DX!(0)), VARSEG(DX!(0)), VARPTR(SRT!(0)),

```

```

        VARSEG(SRT!(0)), VARPTR(FRQ!(0)), VARSEG(FRQ!(0)),
        VARPTR(MAG!(0)), VARSEG(MAG!(0)), NMAX)

PRINT : PRINT "Evaluating the spectrum ..."
FOR I = 1 TO ND2
    TEMP!(I) = MAG!(I)
    IF TEMP!(I) > MAX! THEN MAX! = TEMP!(I)
NEXT
MAX! = THR * MAX! / 100

REM Sort the spectrum and test for chatter.
FOR I = 1 TO ND2
    K = 1
    FOR J = 2 TO ND2
        IF TEMP!(J) > TEMP!(K) THEN K = J
    NEXT
    SRT!(I) = TEMP!(K): IF SRT!(I) < MAX! THEN EXIT FOR
    TEMP!(K) = 0
    J = CINT(F!(K) / FR!): J = CINT(ND2 * (J * FR!) / F!(ND2))
    IF F!(K) >= F!(J) - FD! AND F!(K) <= F!(J) + FD! THEN
        FC! = 0
    ELSE
        FC! = F!(K): EXIT FOR
    END IF
NEXT

REM Plot the spectrum.
XL1$ = "FREQ (Hz)"
YL1$ = "  N =" + STR$(NT)
YL1$ = YL1$ + "  SPEED =" + STR$(CLNG(ACSPD!))
YL1$ = YL1$ + "  Ft =" + STR$(CLNG(FT!))
YL1$ = YL1$ + "  Fc =" + STR$(CLNG(FC!))
CALL PLOT1(F!(), MAG!(), ND2)
SCREEN 0

REM Save the TIME DATA to disk as a BINARY file.
PRINT : PRINT "Save the TIME DATA as a BINARY file? [Y/*N] "
CALL GETKEY(A$): IF A$ <> "Y" THEN GOTO C9
PRINT : REDIM FILE$(1)
PRINT "Enter the BINARY file name for "; TITLE$(0);
INPUT ": ", FILE$(0): IF FILE$(0) = "" THEN GOTO D9
PRINT : CALL WRITE.BINARY(FILE$(0), D0!(), CNT&, DT!)

C9: ERASE TIME!, D0!, DX!, F!, SRT!, FRQ!, MAG!, TEMP!
    NMAX = MAXTEMP
PRINT : PRINT "Continue the CHATTER ANALYSIS module? [*Y/N] ";
CALL GETKEY(A$): IF A$ = "N" THEN GOTO START ELSE GOTO C1

CASE 4

DACHAN = 0: VOLTS! = 0
S1: CLS : VIEW PRINT 1 TO 23

```

```

LOCATE 7, 21: PRINT "
LOCATE 8, 21: PRINT "
LOCATE 9, 21: PRINT "
LOCATE 11, 22: PRINT "[1] READ VOLTAGES FROM A/D CHANNELS"
LOCATE 12, 22: PRINT "[2] WRITE VOLTAGE TO D/A CHANNEL"
LOCATE 13, 22: PRINT "[3] EXIT THE SYSTEM TEST MODULE"
LOCATE 15, 22: PRINT "          Selection? [1-3] "
CALL GETKEY(A$): C = VAL(A$)
IF C < 1 OR C >= 3 THEN GOTO START

IF C = 2 THEN
  CLS : LOCATE 11, 22
  PRINT "Select D/A channel 0 or 1: (";
  PRINT DACHAN; : INPUT ") ", A$
  A! = VAL(A$): IF A! < 0 OR A! > 1 THEN GOTO S1
  IF A$ <> "" THEN DACHAN = A!
  LOCATE 13, 22
  INPUT "Enter D/A voltage: ( MAX +/- 10 ) ", VOLTS!
  IF ABS(VOLTS!) > 10 THEN GOTO S1
END IF

CALL STOP.AND.CLEAR
IF C = 1 THEN LOCATE 17, 18 ELSE LOCATE 15, 18
PRINT "Press RETURN to stop--ready to begin? [*Y/N] "
CALL GETKEY(A$): IF A$ = "N" THEN GOTO S1
IF C = 2 THEN CALL WRITE.DA(DACHAN, VOLTS!)
CLS : LOCATE 25, 1
PRINT "          A/D 0 (0)          A/D 1 (2)";
PRINT "          A/D 2 (4)          A/D 3 (6)"

REM Data reading loop.
S7: CALL READ.AD(0, V0!): CALL READ.AD(1, V1!)
  CALL READ.AD(2, V2!): CALL READ.AD(3, V3!)
  PRINT USING "#####.###"; V0!; V1!; V2!; V3!
  A$ = INKEY$: IF A$ = CHR$(13) OR A$ = CHR$(27) THEN GOTO S9
  GOTO S7
S9: PRINT : PRINT "          Continue? [*Y/N] ";
  CALL GETKEY(A$): IF A$ = "N" THEN GOTO S1
  PRINT : PRINT : GOTO S7

CASE 5

REDIM TITLE$(1), SCL!(1)
I = 0: J = 0: LIM = 1: NUM$ = "": ND2 = NMAX / 2: SCL!(0) = 1
P1: CLS : VIEW PRINT 1 TO 25: TW! = 0
PRINT "
PRINT "
PRINT "
PRINT "
PRINT "
PRINT "Plot a TRANSFER FUNCTION file? [Y/*N] " : CALL GETKEY(FQ$)
IF FQ$ = "Y" THEN
  PRINT

```



```

INPUT "Enter the TF file name (RETURN to exit): ", FILE$
IF FILE$ = "" THEN GOTO P9
PRINT : PRINT "Reading "; FILE$; " ..."
OPEN FILE$ FOR INPUT AS #1
  CNT& = 0
  DO WHILE NOT EOF(1)
    CNT& = CNT& + 1: LINE INPUT #1, A$
  LOOP
CLOSE #1
REDIM F!(CNT&), RE!(CNT&), IM!(CNT&)
OPEN FILE$ FOR INPUT AS #1
  I = 0
  DO WHILE NOT EOF(1)
    I = I + 1: INPUT #1, F!(I), RE!(I), IM!(I)
  LOOP
CLOSE #1
DF! = ABS(F!(4) - F!(3))
XL1$ = "FREQ (Hz)": YL1$ = "REAL TF"
XL2$ = XL1$: YL2$ = "IMAG TF"
LIM = 1: CALL PLOT2(F!(), RE!(), I, F!(), IM!(), I)
SCREEN 0: CLS : GOTO P9
END IF

NCHAN = 1: PRINT
PRINT "Process an ASCII or BINARY data file? [A/*B] "
CALL GETKEY(P$): IF P$ <> "A" THEN P$ = "B"

IF P$ = "A" THEN
  PRINT
  INPUT "Enter the ASCII file name (RETURN to Exit): ", TITLE$(0)
  IF TITLE$(0) = "" THEN GOTO P9
  PRINT : PRINT "Reading "; TITLE$(0); " ..."
  OPEN TITLE$(0) FOR INPUT AS #1
    CNT& = 0
    DO WHILE NOT EOF(1)
      CNT& = CNT& + 1: LINE INPUT #1, A$
    LOOP
  CLOSE #1
  REDIM TIME!(CNT&), DO!(CNT&)
  OPEN TITLE$(0) FOR INPUT AS #1
    I = 0
    DO WHILE NOT EOF(1)
      I = I + 1: INPUT #1, TIME!(I), DO!(I)
    LOOP
  CLOSE #1
  DT! = ABS(TIME!(4) - TIME!(3)): TW! = TIME!(1)
END IF

IF P$ = "B" THEN
  PRINT : TD$ = "Y"
  INPUT "Enter the BINARY file name (RETURN to Exit): ",
    TITLE$(0)

```

```

IF TITLE$(0) = "" THEN GOTO P9
PRINT : PRINT "Reading "; TITLE$(0); " ..."
OPEN TITLE$(0) FOR BINARY ACCESS READ AS #1
  GET #1, , CNT&: GET #1, , DT!
  REDIM TIME!(CNT&), DO!(CNT&)
  FOR I = 1 TO CNT&
    TIME!(I) = (I - 1) * DT!
    GET #1, , DO!(I)
  NEXT
CLOSE #1
IF CNT& = 0 OR DT! = 0 THEN BEEP: KILL TITLE$(0): GOTO P9
END IF

REM Plot the TIME data only.
P3: GOSUB PX1

REM Calculate FFT's and plot TIME data and SPECTRUM.
IF TD$ <> "Y" THEN GOSUB FX1

PRINT
PRINT "Do you want to PLOT or FFT the DATA again? [Y/*N] "
CALL GETKEY(A$)
IF A$ = "Y" THEN
  IF P$ = "B" THEN TW! = 0
  GOTO P3
END IF

REM Save TIME WINDOW DATA to an ASCII file.
IF P$ = "B" THEN
  PRINT : PRINT "Save TIME WINDOW DATA to an ASCII file? [Y/*N] "
  CALL GETKEY(A$): IF A$ <> "Y" THEN GOTO P9
  IF CNT& < NMAX THEN NS = 1: ND = CNT&: GOTO P5
  ET! = INT(((CNT& * DT!) - NMAX * DT!) * 10 ^ 2) / 10 ^ 2
  PRINT : PRINT "Select the START of the TIME WINDOW: ";
  PRINT "( *0 to"; ET!; : PRINT "sec ) "; : INPUT "", TW!
  IF TW! < 0 OR TW! > ET! THEN TW! = 0
  IF TW! = 0 THEN TW! = DT!
  NS = CINT(TW! / DT!): ND = NS + NMAX - 1
P5: PRINT
  PRINT "Enter the ASCII file name for "; TITLE$(0);
  INPUT ": ", FILE$: IF FILE$ = "" THEN GOTO P9
  PRINT : PRINT "Writing "; FILE$: " ..."
  OPEN FILE$ FOR OUTPUT AS #1
  FOR I = 1 TO CNT&
    IF I >= NS AND I <= ND THEN
      PRINT #1, USING "#.#####^ ^ ^ ^ "; TIME!(I); DO!(I)
    END IF
  NEXT
  CLOSE #1
END IF

P9: CNT& = 0: ERASE TIME!, DO!, F!, RE!, IM!

```

```
PRINT : PRINT "Continue the DATA PROCESSING module? [*Y/N] ";
CALL GETKEY(A$): IF A$ = "N" THEN GOTO START ELSE GOTO P1
```

CASE 6

```
M1: CLS : VIEW PRINT 1 TO 25
```

```
PRINT "
PRINT "          PCDATA SETUP MODULE          "
PRINT"
PRINT
PRINT "PcData requires an 80286 computer with CGA or EGA"
PRINT "graphics, a math coprocessor, and a DT-2818 data"
PRINT "acquisition board at the base address &H2EC configured to"
PRINT "+/- 10 volts. DMA channel 1 is used, with the data stored"
PRINT "in memory page 4. The printer port is assumed to be LPT1."
PRINT : PRINT
PRINT "Enter the hex BASE ADDRESS for the DT-2818: ( &H";
PRINT HEX$(BASE.ADR); " ) "; : INPUT "", A$
IF A$ = "" THEN BASE.ADR = BASE.ADR ELSE BASE.ADR = VAL(A$)
DATA.REG = BASE.ADR
COM.REG = BASE.ADR + 1: STAT.REG = BASE.ADR + 1
PRINT
PRINT "Select DMA CHANNEL 1 or DMA CHANNEL 3: (";
PRINT DMACHAN; ") "; : INPUT "", A$
IF A$ = "" THEN DMACHAN = DMACHAN ELSE DMACHAN = VAL(A$)
IF DMACHAN <> 1 AND DMACHAN <> 3 THEN BEEP: GOTO M9
IF DMACHAN = 1 THEN
  DMACHAN = &H1: DMACOM = &H45
  BASEREG = &H2: COUNTREG = &H3: PAGEREG = &H83
END IF
IF DMACHAN = 3 THEN
  DMACHAN = &H3: DMACOM = &H47
  BASEREG = &H6: COUNTREG = &H7: PAGEREG = &H82
END IF
PRINT
PRINT "Enter the SIZE for the Fast Fourier Transforms: (";
PRINT NMAX; ") "; : INPUT "", A$
IF A$ = "" THEN NMAX = NMAX ELSE NMAX = VAL(A$)
IF NMAX <> 1024 AND NMAX <> 2048 AND NMAX <> 4096
  AND NMAX <> 8192 THEN
  NMAX = 4096: BEEP: GOTO M9
END IF
```

```
M9: PRINT : PRINT "Continue the PCDATA SETUP module? [Y/*N] ";
CALL GETKEY(A$): IF A$ = "Y" THEN GOTO M1 ELSE GOTO START
```

CASE 7

```
REM Terminate the program.
```

```
CLS : END
```

```
END SELECT
```

REM \*\*\*\*\* PLOT TIME DATA ONLY SUBROUTINE \*\*\*\*\*

```

PX1:
PRINT : PRINT "Plot the TIME DATA only? [Y/*N] "
TD$ = "": CALL GETKEY(TD$): IF TD$ <> "Y" THEN GOTO PX9
NN = CNT&: XL1$ = "TIME (sec)"
YL1$ = TITLE$(0): CALL PLOT1(TIME!(), DO!(), NN)
IF NCHAN = 2 THEN
  YL1$ = TITLE$(1): CALL PLOT1(TIME!(), D1!(), NN)
END IF
IF NCHAN = 3 THEN
  YL1$ = TITLE$(1): CALL PLOT1(TIME!(), D1!(), NN)
  YL1$ = TITLE$(2): CALL PLOT1(TIME!(), D2!(), NN)
END IF
IF NCHAN = 4 THEN
  YL1$ = TITLE$(1): CALL PLOT1(TIME!(), D1!(), NN)
  YL1$ = TITLE$(2): CALL PLOT1(TIME!(), D2!(), NN)
  YL1$ = TITLE$(3): CALL PLOT1(TIME!(), D3!(), NN)
END IF
SCREEN 0: CLS
PX9: RETURN

```

REM \*\*\*\*\* COMPUTE FFT & PLOT DATA AND SPECTRUM SUBROUTINE \*\*\*\*\*

```

FX1:
IF CNT& < NMAX THEN GOTO FX9
S1$ = "": S2$ = "": IF NCHAN > 1 THEN S1$ = "S": S2$ = "s"
PRINT : PRINT "Compute FAST FOURIER TRANSFORM"; S1$;
PRINT " of the DATA? [*Y/N] "
CALL GETKEY(A$): IF A$ = "N" THEN GOTO FX9
REDIM T!(NMAX), X0!(NMAX), TEMP!(NMAX): NUM$ = "one"
IF NCHAN = 2 THEN REDIM X1!(NMAX): NUM$ = "two"
IF NCHAN = 3 THEN REDIM X1!(NMAX), X2!(NMAX): NUM$ = "three"
IF NCHAN = 4 THEN
  REDIM X1!(NMAX), X2!(NMAX), X3!(NMAX): NUM$ = "four"
END IF
ET! = INT(((CNT& * DT!) - NMAX * DT!) * 10 ^ 2) / 10 ^ 2
PRINT
PRINT "TIME STEP:"; CINT(DT! * 10 ^ 6) / 10 ^ 6; "sec."
PRINT "TIME WINDOW for"; NMAX; "point FFT:";
PRINT CINT((NMAX * DT!) * 10 ^ 2) / 10 ^ 2; "sec."
IF CNT& = NMAX THEN NS = 1: ND = NMAX: GOTO FX3
PRINT : PRINT "Select the START of the TIME WINDOW: ";
PRINT "( *0 to"; ET!; : PRINT "sec ) "; : INPUT "", TW!
IF TW! < 0 OR TW! > ET! THEN TW! = 0
IF TW! = 0 THEN TW! = DT!
PRINT : NS = CINT(TW! / DT!): ND = NS + NMAX - 1: NN = 0

REM Apply the TIME WINDOW to the DATA.
FX3: J = 0
FOR I = 1 TO CNT&
  IF I >= NS AND I <= ND THEN

```

```

J = J + 1: T!(J) = TIME!(I): X0!(J) = D0!(I)
IF NCHAN = 2 THEN
  X1!(J) = D1!(I)
END IF
IF NCHAN = 3 THEN
  X1!(J) = D1!(I): X2!(J) = D2!(I)
END IF
IF NCHAN = 4 THEN
  X1!(J) = D1!(I): X2!(J) = D2!(I): X3!(J) = D3!(I)
END IF
END IF
NEXT

DF! = 1 / (J * DT!): IF J <> NMAX THEN BEEP: GOTO START
REDIM F!(ND2), RE!(ND2), IM!(ND2)
REDIM MAG0!(ND2), MAG1!(ND2), MAG2!(ND2), MAG3!(ND2)

REM Compute frequency range.
FOR I = 1 TO ND2: F!(I) = I * DF!: NEXT
PRINT "FREQUENCY RESOLUTION:"; CINT(DF! * 10 ^ 2) / 10 ^ 2; "Hz."
PRINT "FREQUENCY LIMIT:"; CINT(1 / (DT! * 2)); "Hz."

REM Compute the Fast Fourier Transform(s).
PRINT : PRINT "Computing "; NUM$; J;
PRINT "point Fast Fourier Transform"; S2$; " ...";
FOR I = 1 TO J: TEMP!(I) = X0!(I): NEXT
CALL FFT(VARPTR(TEMP!(0)), VARSEG(TEMP!(0)), VARPTR(RE!(0)),
  VARSEG(RE!(0)), VARPTR(IM!(0)), VARSEG(IM!(0)),
  VARPTR(MAG0!(0)), VARSEG(MAG0!(0)), J)
IF NCHAN = 2 THEN
  FOR I = 1 TO J: TEMP!(I) = X1!(I): NEXT
  CALL FFT(VARPTR(TEMP!(0)), VARSEG(TEMP!(0)), VARPTR(RE!(0)),
    VARSEG(RE!(0)), VARPTR(IM!(0)), VARSEG(IM!(0)),
    VARPTR(MAG1!(0)), VARSEG(MAG1!(0)), J)
END IF
IF NCHAN = 3 THEN
  FOR I = 1 TO J: TEMP!(I) = X1!(I): NEXT
  CALL FFT(VARPTR(TEMP!(0)), VARSEG(TEMP!(0)), VARPTR(RE!(0)),
    VARSEG(RE!(0)), VARPTR(IM!(0)), VARSEG(IM!(0)),
    VARPTR(MAG1!(0)), VARSEG(MAG1!(0)), J)
  FOR I = 1 TO J: TEMP!(I) = X2!(I): NEXT
  CALL FFT(VARPTR(TEMP!(0)), VARSEG(TEMP!(0)), VARPTR(RE!(0)),
    VARSEG(RE!(0)), VARPTR(IM!(0)), VARSEG(IM!(0)),
    VARPTR(MAG2!(0)), VARSEG(MAG2!(0)), J)
END IF
IF NCHAN = 4 THEN
  FOR I = 1 TO J: TEMP!(I) = X1!(I): NEXT
  CALL FFT(VARPTR(TEMP!(0)), VARSEG(TEMP!(0)), VARPTR(RE!(0)),
    VARSEG(RE!(0)), VARPTR(IM!(0)), VARSEG(IM!(0)),
    VARPTR(MAG1!(0)), VARSEG(MAG1!(0)), J)
  FOR I = 1 TO J: TEMP!(I) = X2!(I): NEXT
  CALL FFT(VARPTR(TEMP!(0)), VARSEG(TEMP!(0)), VARPTR(RE!(0)),

```

```

        VARSEG(RE!(0)), VARPTR(IM!(0)), VARSEG(IM!(0)),
        VARPTR(MAG2!(0)), VARSEG(MAG2!(0)), J)
    FOR I = 1 TO J: TEMP!(I) = X3!(I): NEXT
    CALL FFT(VARPTR(TEMP!(0)), VARSEG(TEMP!(0)), VARPTR(RE!(0)),
        VARSEG(RE!(0)), VARPTR(IM!(0)), VARSEG(IM!(0)),
        VARPTR(MAG3!(0)), VARSEG(MAG3!(0)), J)
END IF
ERASE TEMP!, RE!, IM!

REM Plot the TIME DATA and SPECTRUM for each channel.
XL1$ = "TIME (sec)": XL2$ = "FREQ (Hz)"
YL1$ = TITLE$(0): YL2$ = TITLE$(0) + "-SPECTRUM"
CALL PLOT2(T!(), X0!(), J, F!(), MAG0!(), ND2)
ERASE X0!, MAG0!
IF NCHAN = 2 THEN
    YL1$ = TITLE$(1): YL2$ = TITLE$(1) + "-SPECTRUM"
    CALL PLOT2(T!(), X1!(), J, F!(), MAG1!(), ND2)
    ERASE X1!, MAG1!
END IF
IF NCHAN = 3 THEN
    YL1$ = TITLE$(1): YL2$ = TITLE$(1) + "-SPECTRUM"
    CALL PLOT2(T!(), X1!(), J, F!(), MAG1!(), ND2)
    YL1$ = TITLE$(2): YL2$ = TITLE$(2) + "-SPECTRUM"
    CALL PLOT2(T!(), X2!(), J, F!(), MAG2!(), ND2)
    ERASE X1!, MAG1!, X2!, MAG2!
END IF
IF NCHAN = 4 THEN
    YL1$ = TITLE$(1): YL2$ = TITLE$(1) + "-SPECTRUM"
    CALL PLOT2(T!(), X1!(), J, F!(), MAG1!(), ND2)
    YL1$ = TITLE$(2): YL2$ = TITLE$(2) + "-SPECTRUM"
    CALL PLOT2(T!(), X2!(), J, F!(), MAG2!(), ND2)
    YL1$ = TITLE$(3): YL2$ = TITLE$(3) + "-SPECTRUM"
    CALL PLOT2(T!(), X3!(), J, F!(), MAG3!(), ND2)
    ERASE X1!, MAG1!, X2!, MAG2!, X3!, MAG3!
END IF
ERASE T!, F!: SCREEN 0: CLS
FX9: RETURN

REM ***** CHECK DT-2818 ERROR SUBROUTINE *****
SUB CHECK.ERROR

    WAIT STAT.REG, WRITE.WAIT, WRITE.WAIT
    WAIT STAT.REG, COM.WAIT
    Status = INP(STAT.REG)
    IF (Status AND &H80) THEN
        BEEP
        PRINT : PRINT "*** DT-2818 ERROR! ***": PRINT
        END
    END IF

END SUB

```

```

REM ***** CURVE PLOTTING SUBROUTINE *****
SUB CURVE (X!(), Y!(), NP) STATIC

```

```

  SHARED XM!, XL!, YM!, YL!, XD, YD, YPL, YB, YE, CUH
  SHARED SC$, XL1$, YL1$, XL2$, YL2$, XLABEL$, YLABEL$
  SHARED OOPS, REDRAW, PLOTCount, LIM, PLOTONE, PLOTTwo

```

```

  IF REDRAW THEN GOTO CONTINUE

```

```

REM Find the plot limits (X axis values must be ascending).

```

```

  XL! = X!(1): XM! = X!(NP): YL! = 1E+30: YM! = -1E+30
  FOR I = 1 TO NP
    IF Y!(I) > YM! THEN YM! = Y!(I)
    IF Y!(I) < YL! THEN YL! = Y!(I)
  NEXT
  XD = 5: YD = 4

```

```

CONTINUE:

```

```

  CONST XB = 70, XE = 600, XPL = XE - XB
  XR! = ABS(XM! - XL!): IF XR! = 0 THEN XR! = 1
  TESTX! = XR! / XD: XFAC! = XPL / XR!
  XF$ = "##.#####": XCF$ = "##.#####"
  YR! = ABS(YM! - YL!): IF YR! = 0 THEN YR! = 1
  TESTY! = YR! / YD: YFAC! = YPL / YR!
  YF$ = "##.#####": YCF$ = "##.#####"

```

```

REM Find the format for the axis divisions.

```

```

  IF TESTX! >= .0001 THEN XF$ = "###.#####": XCF$ = "###.#####"
  IF TESTY! >= .0001 THEN YF$ = "###.#####": YCF$ = "###.#####"
  IF TESTX! >= .001 THEN XF$ = "###.###": XCF$ = "###.###"
  IF TESTY! >= .001 THEN YF$ = "###.###": YCF$ = "###.###"
  IF TESTX! >= .01 THEN XF$ = "###.##": XCF$ = "###.##"
  IF TESTY! >= .01 THEN YF$ = "###.##": YCF$ = "###.##"
  IF TESTX! >= 1 THEN XF$ = "###.#": XCF$ = "###.##"
  IF TESTY! >= 1 THEN YF$ = "###.#": YCF$ = "###.##"
  IF TESTX! >= 10 THEN XF$ = "#####": XCF$ = "#####.##"
  IF TESTY! >= 10 THEN YF$ = "#####": YCF$ = "#####.##"
  IF TESTX! >= 100 THEN XF$ = "#####": XCF$ = "#####.##"
  IF TESTY! >= 100 THEN YF$ = "#####": YCF$ = "#####.##"
  IF TESTX! >= 1000 THEN XF$ = "#####": XCF$ = "#####.##"
  IF TESTY! >= 1000 THEN YF$ = "#####": YCF$ = "#####.##"
  IF TESTX! >= 10000 THEN XF$ = "##.#####": XCF$ = "##.#####"
  IF TESTY! >= 10000 THEN YF$ = "##.#####": YCF$ = "##.#####"

```

```

REM Set the screen type and set up the plotting area.

```

```

  IF SC$ = "E" THEN
    SCREEN 9
    VIEW SCREEN (0, YB - CUH)-(639, YE + (349 - YE)): CLS
  ELSE
    SCREEN 2
    VIEW SCREEN (0, YB - CUH)-(639, YE + (199 - YE)): CLS
  END IF

```

REM Draw the border.

```
LINE (XB, YE)-(XE, YB), , B
```

REM Draw X axis divisions.

```
FOR I = 0 TO XD
```

```
LINE (XB + I * XPL / XD, YB)-(XB + I * XPL / XD, YE)
, , , &HCCCC
```

```
LOCATE YE / CUH + 1.5, INT((XB + I * XPL / XD) / 8) - 2
PRINT USING XF$; XL! + XR! / XD * I * SGN(XM! - XL!);
```

```
NEXT
```

```
LOCATE YE / CUH + 2.5, 39: PRINT XLABEL$;
```

REM Draw Y axis divisions.

```
FOR I = 0 TO YD
```

```
LINE (XB, YE - I * YPL / YD)-(XE, YE - I * YPL / YD)
, , , &HCCCC
```

```
LOCATE INT((YE - I * YPL / YD) / CUH) + 1, 8 / LEN(YF$)
PRINT USING YF$; YL! + YR! / YD * I * SGN(YM! - YL!)
```

```
NEXT
```

```
LOCATE YB / CUH, XE / 8 - LEN(YLABEL$): PRINT YLABEL$
```

REM Plot the curve.

```
FOR I = 1 TO NP
```

```
IF X!(I) >= XL! THEN NL = I: EXIT FOR
```

```
NEXT
```

```
IF NL > 1 THEN NL = NL - 1
```

```
VIEW SCREEN (XB, YB)-(XE, YE)
```

```
FOR I = NL TO NP
```

```
IF INKEY$ = CHR$(27) THEN EXIT FOR
```

```
XP! = XB + (X!(I) - XL!) * XFAC!
```

```
YP! = YE - (Y!(I) - YL!) * YFAC!
```

```
IF I > NL AND ABS(YP!) < 351 AND ABS(XP!) < 641 THEN
```

```
LINE (XP!, YP!)-(XP1!, YP1!)
```

```
END IF
```

```
XP1! = XP!: YP1! = YP!
```

```
IF X!(I) > XM! THEN EXIT FOR
```

```
NEXT
```

```
TOP = I - 1
```

REM Present the Plotting Options.

A0:

```
IF LIM = 0 THEN EXIT SUB
```

```
VIEW: U = YB / CUH: W = 1: REDRAW = 0: OOPS = 0: HELP = 1
```

```
LOCATE 25, 1: PRINT " F1=LIMITS F2=LABEL F3=REPLOT F4=";
```

```
PRINT "SINGLE F5=CURSOR F6=MATH F7=PRINT ESC=CONTINUE ";
```

A1:

```
AN$ = INKEY$: IF AN$ = "" THEN GOTO A1
```

REM Press ESCAPE to quit the plot.

```
IF AN$ = CHR$(&H1B) THEN
```



```

LOCATE U, W: PRINT " "
VIEW: OOPS = 0: REDRAW = 0: EXIT SUB
END IF

```

REM Press RETURN to toggle the help line.

```

IF AN$ = CHR$(&HD) THEN
  IF HELP = 1 THEN
    LOCATE 25, 1: PRINT STRING$(80, " ");
    HELP = 0: GOTO A1
  END IF
  IF HELP = 0 THEN
    LOCATE 25, 1: PRINT " F1=LIMITS F2=LABEL F3=REPLOTT F4=";
    PRINT "SINGLE F5=CURSOR F6=MATH F7=PRINT ESC=CONTINUE ";
    HELP = 1: GOTO A1
  END IF
END IF

```

REM Press F1 key to change plot limits.

```

IF MID$(AN$, 2, 1) = ";" THEN
  LOCATE 25, 1: PRINT STRING$(80, " ");
  LOCATE 25, 1: PRINT " ";
  PRINT "Press RETURN to keep the current value";
A2: U = YB / CUH: W = 1
  LOCATE U, W: PRINT STRING$(79, " ")
  LOCATE U, W: INPUT "XLOW = ", A$
  A! = VAL(A$): IF A$ <> "" THEN XL! = A!
  LOCATE U, W: PRINT STRING$(79, " ")
  LOCATE U, W: INPUT "XMAX = ", A$
  A! = VAL(A$): IF A$ <> "" THEN XM! = A!
  IF XM! <= XL! THEN BEEP: GOTO A2
  LOCATE U, W: PRINT STRING$(79, " ")
  LOCATE U, W: INPUT "XDIV = ", A$
  A! = VAL(A$): IF A$ <> "" THEN XD = A!
A3: LOCATE U, W: PRINT STRING$(79, " ")
  LOCATE U, W: INPUT "YLOW = ", A$
  A! = VAL(A$): IF A$ <> "" THEN YL! = A!
  LOCATE U, W: PRINT STRING$(79, " ")
  LOCATE U, W: INPUT "YMAX = ", A$
  A! = VAL(A$): IF A$ <> "" THEN YM! = A!
  IF YM! <= YL! THEN BEEP: GOTO A3
  LOCATE U, W: PRINT STRING$(79, " ")
  LOCATE U, W: INPUT "YDIV = ", A$
  A! = VAL(A$): IF A$ <> "" THEN YD = A!
  IF XD <= 0 OR XD > 10 THEN XD = 1
  IF YD <= 0 OR YD > 10 THEN YD = 1
  REDRAW = 1: EXIT SUB
END IF

```

REM Press F2 key to label the plot.

```

IF MID$(AN$, 2, 1) = "<" THEN
  LOCATE 25, 1: PRINT " ";
  PRINT "ARROW KEYS = ";

```

```

PRINT "Position the Cursor          ";
REDIM CSR(8 * CUH): LOCATE 2, 78: PRINT " "
GET (616, CUH)-(624, 2 * CUH), CSR: LOCATE 2, 78: PRINT " "
U = YB / CUH: W = 1: LOCATE U, W: PRINT " "
W = XB / 8 + 1
PUT ((W - 1) * 8, (U - 1) * CUH), CSR
A5: A$ = INKEY$: IF A$ = "" THEN GOTO A5
IF A$ = CHR$(13) THEN GOTO A5
IF A$ = CHR$(9) THEN GOTO A5
IF MID$(A$, 2, 1) = ";" THEN GOTO A5
IF MID$(A$, 2, 1) = "<" THEN GOTO A5
IF MID$(A$, 2, 1) = ">" THEN GOTO A5
REM ESC to mask the cursor and stop labeling.
IF A$ = CHR$(&H1B) THEN
    PUT ((W - 1) * 8, (U - 1) * CUH), CSR
    GOTO A0
END IF
IF A$ = CHR$(8) THEN
    REM BACKSPACE key.
    A$ = " ": PUT ((W - 1) * 8, (U - 1) * CUH), CSR
    W = W - 1
    IF W < 1 AND U = 1 THEN W = 1
    IF W < 1 AND U <> 1 THEN W = 79: U = U - 1
    LOCATE U, W: PRINT A$;
    PUT ((W - 1) * 8, (U - 1) * CUH), CSR
    GOTO A5
END IF
REM Extended ARROW keys.
IF MID$(A$, 2, 1) = "H" THEN
    REM UP arrow.
    PUT ((W - 1) * 8, (U - 1) * CUH), CSR
    U = U - 1: IF U < 1 THEN U = 24
    PUT ((W - 1) * 8, (U - 1) * CUH), CSR
    GOTO A5
END IF
IF MID$(A$, 2, 1) = "K" THEN
    REM LEFT arrow.
    PUT ((W - 1) * 8, (U - 1) * CUH), CSR
    W = W - 1
    IF W < 1 AND U = 1 THEN W = 1
    IF W < 1 AND U <> 1 THEN W = 79: U = U - 1
    PUT ((W - 1) * 8, (U - 1) * CUH), CSR
    GOTO A5
END IF
IF MID$(A$, 2, 1) = "M" THEN
    REM RIGHT arrow.
    PUT ((W - 1) * 8, (U - 1) * CUH), CSR
    W = W + 1
    IF W > 79 AND U = 24 THEN W = 79
    IF W > 79 AND U <> 24 THEN W = 1: U = U + 1
    PUT ((W - 1) * 8, (U - 1) * CUH), CSR
    GOTO A5

```

```

END IF
IF MID$(A$, 2, 1) = "P" THEN
  REM DOWN arrow.
  PUT ((W - 1) * 8, (U - 1) * CUH), CSR
  U = U + 1: IF U > 24 THEN U = 1
  PUT ((W - 1) * 8, (U - 1) * CUH), CSR
  GOTO A5
END IF
IF MID$(A$, 2, 1) = "R" OR MID$(A$, 2, 1) = "S" THEN
  REM INSERT and DELETE keys not used.
  GOTO A5
END IF
IF MID$(A$, 2, 1) = "I" THEN
  REM PAGE UP
  A$ = " ": PUT ((W - 1) * 8, (U - 1) * CUH), CSR
  U = 1: PUT ((W - 1) * 8, (U - 1) * CUH), CSR
  GOTO A5
END IF
IF MID$(A$, 2, 1) = "Q" THEN
  REM PAGE DOWN
  A$ = " ": PUT ((W - 1) * 8, (U - 1) * CUH), CSR
  U = 24: PUT ((W - 1) * 8, (U - 1) * CUH), CSR
  GOTO A5
END IF
IF MID$(A$, 2, 1) = "G" THEN
  REM HOME
  A$ = " ": PUT ((W - 1) * 8, (U - 1) * CUH), CSR
  U = 1: W = 1: PUT ((W - 1) * 8, (U - 1) * CUH), CSR
  GOTO A5
END IF
IF MID$(A$, 2, 1) = "O" THEN
  REM END
  A$ = " ": PUT ((W - 1) * 8, (U - 1) * CUH), CSR
  U = 24: W = 78
END IF
REM Move the cursor.
LOCATE U, W: W = W + 1: IF W > 79 THEN W = 1: U = U + 1
IF U > 24 THEN U = 24
PUT ((W - 1) * 8, (U - 1) * CUH), CSR
PRINT A$; : GOTO A5: REM Print the label character.
END IF

```

```

REM Press F3 key to replot.
IF MID$(AN$, 2, 1) = "=" THEN
  OOPS = 1: REDRAW = 0: EXIT SUB
END IF

```

```

REM Press F4 key to plot on single screen.
IF MID$(AN$, 2, 1) = ">" AND PLOT TWO THEN
  OOPS = 0: REDRAW = 0
  TEMPX$ = XL1$: TEMPY$ = YL1$
  IF PLOT COUNT = 1 THEN XL1$ = XLABEL$: YL1$ = YLABEL$

```

```

IF PLOTCOUNT = 2 THEN XL1$ = XL2$: YL1$ = YL2$
CALL PLOT1(X!(), Y!(), NP)
PLOTONE = 0: PLOTTWO = 1: OOPS = 1: REDRAW = 0
XL1$ = TEMPX$: YL1$ = TEMPY$
EXIT SUB
END IF

```

REM Press F5 key for the data value cursor.

```

IF MID$(AN$, 2, 1) = "?" THEN
  LOCATE 25, 1
  PRINT " ";
  PRINT "CTRL ARROW KEYS = Fast Cursor ";
  REDIM CLN(24 * CUH): LOCATE 2, 78: PRINT "|";
  LOCATE 3, 78: PRINT "|": LOCATE 4, 78: PRINT "|";
  GET (616, CUH)-(624, 4 * CUH), CLN: LOCATE 2, 78: PRINT " ";
  LOCATE 3, 78: PRINT " ": LOCATE 4, 78: PRINT " ";
  I = NL: GOTO A4X
A4: A$ = INKEY$: IF A$ = "" THEN GOTO A4
  IF A$ = CHR$(&H1B) THEN
    REM Press ESCAPE to quit cursor option.
    PUT (XP!, YP!), CLN
    LOCATE YE / CUH + 2.5, 1
    PRINT STRING$(40 - LEN(XLABEL$) / 2, " "); : GOTO A0
  END IF
  A$ = MID$(A$, 2, 1)
  IF A$ = "K" OR A$ = "s" OR A$ = "M" OR A$ = "t" THEN
    PUT (XP!, YP!), CLN
    REM LEFT arrow.
    IF A$ = "K" THEN INC = -1
    REM CTRL LEFT arrow.
    IF A$ = "s" THEN INC = -(0.01 * NP)
    REM RIGHT arrow.
    IF A$ = "M" THEN INC = 1
    REM CTRL RIGHT arrow.
    IF A$ = "t" THEN INC = (0.01 * NP)
    I = I + INC
    IF I < NL THEN I = TOP
    IF I > TOP THEN I = NL
A4X: LOCATE YE / CUH + 2.5, 1
    PRINT "X = "; : PRINT USING XCF$; X!(I);
    PRINT " Y = "; : PRINT USING YCF$; Y!(I);
    XP! = XB + (X!(I) - XL!) * XFAC! - 3
    YP! = YE - (Y!(I) - YL!) * YFAC! - 1.5 * CUH
    IF YP! < YB THEN YP! = YB - 1.5 * CUH
    IF YP! < 0 THEN YP! = YB
    IF YP! > YE THEN YP! = YE - 1.5 * CUH
    PUT (XP!, YP!), CLN
    GOTO A4
  END IF
  GOTO A4
END IF

```

```

REM Press F6 key for math options.
IF MID$(AN$, 2, 1) = "@" THEN
  LOCATE 25, 1
  PRINT " S=SMOOTH      D=DIVIDE BY -W^2";
  PRINT "      X=SCALE X AXIS  Y=SCALE Y AXIS      ";
A6: A$ = INKEY$: IF A$ = "" THEN GOTO A6
IF A$ = CHR$(&H1B) THEN GOTO A0
IF UCASE$(A$) = "S" THEN
  LOCATE 25, 1: PRINT STRING$(80, " ");
  CALL SMOOTH(1, 5, Y!(), NP)
  OOPS = 0: REDRAW = 1: EXIT SUB
END IF
IF UCASE$(A$) = "D" THEN
  LOCATE 25, 1: PRINT STRING$(80, " ");
  FOR I = 1 TO NP
    Y!(I) = Y!(I) / -((X!(I) * 6.283186) ^ 2)
  NEXT
  OOPS = 0: REDRAW = 1: EXIT SUB
END IF
IF UCASE$(A$) = "X" THEN
  LOCATE 25, 1: PRINT STRING$(80, " ");
  U = YB / CUH: W = 1
  LOCATE U, W: PRINT STRING$(79, " ")
  LOCATE U, W: INPUT "SCALE X AXIS BY: ", A$
  LOCATE U, W: PRINT STRING$(79, " ")
  A! = VAL(A$): IF A$ = "" THEN GOTO A0
  FOR I = 1 TO NP
    X!(I) = X!(I) * A!
  NEXT
  A$ = "": OOPS = 0: REDRAW = 1: EXIT SUB
END IF
IF UCASE$(A$) = "Y" THEN
  LOCATE 25, 1: PRINT STRING$(80, " ");
  U = YB / CUH: W = 1
  LOCATE U, W: PRINT STRING$(79, " ")
  LOCATE U, W: INPUT "SCALE Y AXIS BY: ", A$
  LOCATE U, W: PRINT STRING$(79, " ")
  A! = VAL(A$): IF A$ = "" THEN GOTO A0
  FOR I = 1 TO NP
    Y!(I) = Y!(I) * A!
  NEXT
  A$ = "": OOPS = 0: REDRAW = 1: EXIT SUB
END IF
GOTO A6
END IF

```

```

REM Press F7 to print the plot. Printer must be a LaserJet
REM Series II or an Epson compatible located at LPT1:
IF MID$(AN$, 2, 1) = "A" THEN
  AZ$ = "": PRN$ = "E": DPI$ = ""
  LPT = 1: XS = 1: YS = 1: OT = 1
  LOCATE 25, 1

```

```

PRINT " J=LASERJET II    *E=EPSON";
PRINT "      P=PORTRAIT  *L=LANDSCAPE";
PRINT "      G=GO        ";
DO WHILE AZ$ <> "G"
  CALL GETKEY(AZ$)
  IF AZ$ = "J" THEN
    PRN$ = "L": LOCATE 25, 1
    PRINT " J=LASERJET II          ";
    PRINT "      P=PORTRAIT      L=LANDSCAPE";
  END IF
  IF AZ$ = "E" THEN
    PRN$ = "E": LOCATE 25, 1
    PRINT "                          E=EPSON";
  END IF
  IF AZ$ = "P" THEN
    OT = 0: DPI$ = "100": LOCATE 25, 26
    PRINT "      P=PORTRAIT          ";
  END IF
  IF AZ$ = "L" THEN
    OT = 1: DPI$ = "075": LOCATE 25, 26
    PRINT "                          L=LANDSCAPE";
  END IF
  IF AZ$ = CHR$(27) THEN GOTO A0
  IF PRN$ = "E" THEN
    DPI$ = "": XS = 1: YS = 1
    IF SC$ = "C" THEN XS = 1: YS = 2
  END IF
  REM Make the plot.
  IF AZ$ = "G" AND PRN$ <> "" THEN
    IF PRN$ = "L" THEN
      IF DPI$ = "" THEN DPI$ = "100": OT = 0
      LPRINT CHR$(27) + "*p" + "200" + "Y";
      IF OT = 0 THEN LPRINT CHR$(27) + "*p" + "300" + "X";
      IF OT = 1 THEN LPRINT CHR$(27) + "*p" + "400" + "X";
    END IF
    LOCATE 25, 1: PRINT STRING$(80, " ");
    CALL ScrnDump2(DPI$, LPT, 0, XS, YS, OT)
    IF LPT = -1 THEN
      BEEP: GOTO A0: REM No Printer.
    END IF
    IF PRN$ = "E" AND OT = 0 THEN LPRINT CHR$(13);
    IF PRN$ = "E" AND OT = 1 THEN LPRINT CHR$(12);
    IF PRN$ = "E" THEN LPRINT CHR$(27) + "<";
    IF PRN$ = "L" THEN LPRINT CHR$(27) + "E";
    GOTO A0
  END IF
LOOP
END IF
GOTO A1
END SUB

```

```
REM ***** GET SINGLE KEY INPUT SUBROUTINE *****
SUB GETKEY (A$)
```

```
  A$ = ""
  DO WHILE A$ = "": A$ = INKEY$: LOOP
  A$ = UCASE$(A$)
```

```
END SUB
```

```
REM ***** MAIN MENU SUBROUTINE *****
SUB MENU (A)
```

```
CLS : VIEW PRINT 1 TO 25
LOCATE 6, 21: PRINT "          PcData (ver 1.0)          "
LOCATE 7, 21: PRINT "          "
LOCATE 8, 21: PRINT "          [1] DATA ACQUISITION MODULE          "
LOCATE 9, 21: PRINT "          [2] TRANSFER FUNCTION MODULE          "
LOCATE 10, 21: PRINT "          [3] CHATTER ANALYSIS MODULE          "
LOCATE 11, 21: PRINT "          [4] SYSTEM TEST MODULE          "
LOCATE 12, 21: PRINT "          [5] DATA PROCESSING MODULE          "
LOCATE 13, 21: PRINT "          [6] PCDATA SETUP MODULE          "
LOCATE 14, 21: PRINT "          [7] QUIT THE PROGRAM          "
LOCATE 15, 21: PRINT "          "
LOCATE 16, 21: PRINT "          Selection? [1-7]          "
LOCATE 17, 21: PRINT "          "
LOCATE 18, 21: PRINT "          "
```

```
MX: A$ = INKEY$: A = VAL(A$)
  IF A$ = CHR$(27) THEN CLS : END
  IF A < 1 OR A > 7 THEN GOTO MX
```

```
END SUB
```

```
REM ***** DRAW ONE PLOT ON THE SCREEN SUBROUTINE *****
SUB PLOT1 (X!(), Y1!(), N) STATIC
```

```
  SHARED XM!, XL!, YM!, YL!, XD, YD, YPL, YB, YE, CUH
  SHARED SC$, XL1$, YL1$, XL2$, YL2$, XLABEL$, YLABEL$
  SHARED OOPS, REDRAW, PLOTCOUNT, LIM, PLOTONE, PLOTTWO
```

```
CLS : PLOTONE = 1: PLOTTWO = 0: PLOTCOUNT = 1: YD = 4
```

```
IF SC$ = "C" THEN
G1: CUH = 8: YB = CUH: YE = 165: YPL = (YE - YB)
  XLABEL$ = XL1$: YLABEL$ = YL1$
  CALL CURVE(X!(), Y1!(), N)
  IF REDRAW OR OOPS THEN GOTO G1
END IF
IF SC$ = "E" THEN
G2: CUH = 14: YB = CUH: YE = 303: YPL = (YE - YB)
```

```

XLABEL$ = XL1$: YLABEL$ = YL1$
CALL CURVE(X!(), Y1!(), N)
IF REDRAW OR OOPS THEN GOTO G2
END IF

```

```
END SUB
```

```

REM ***** DRAW TWO PLOTS ON ONE SCREEN SUBROUTINE *****
SUB PLOT2 (X1!(), Y1!(), N1, X2!(), Y2!(), N2) STATIC

```

```

  SHARED XM!, XL!, YM!, YL!, XD, YD, YPL, YB, YE, CUH
  SHARED SC$, XL1$, YL1$, XL2$, YL2$, XLABEL$, YLABEL$
  SHARED OOPS, REDRAW, PLOTCOUNT, LIM, PLOTONE, PLOTTWO

```

```
  YD = 4: PLOTONE = 0: PLOTTWO = 1: PLOTCOUNT = 0
```

```
  IF SC$ = "C" THEN
```

```

K1: CLS : CUH = 8: YB = CUH: YE = 78: YPL = YE - YB
    PLOTCOUNT = 1: XLABEL$ = XL1$: YLABEL$ = YL1$
    CALL CURVE(X1!(), Y1!(), N1)
    IF REDRAW OR OOPS THEN GOTO K1

```

```

K2: CUH = 8: YB = 104: YE = 174: YPL = YE - YB
    PLOTCOUNT = 2: XLABEL$ = XL2$: YLABEL$ = YL2$
    CALL CURVE(X2!(), Y2!(), N2)
    IF OOPS THEN GOTO K1
    IF REDRAW THEN GOTO K2

```

```
  END IF
```

```
  IF SC$ = "E" THEN
```

```

E1: CLS : CUH = 14: YB = CUH: YE = 136: YPL = YE - YB
    PLOTCOUNT = 1: XLABEL$ = XL1$: YLABEL$ = YL1$
    CALL CURVE(X1!(), Y1!(), N1)
    IF REDRAW OR OOPS THEN GOTO E1

```

```

E2: CUH = 14: YB = 182: YE = 304: YPL = YE - YB
    PLOTCOUNT = 2: XLABEL$ = XL2$: YLABEL$ = YL2$
    CALL CURVE(X2!(), Y2!(), N2)
    IF OOPS THEN GOTO E1
    IF REDRAW THEN GOTO E2

```

```
  END IF
```

```
END SUB
```

```

REM ***** READ A TO D IMMEDIATE SUBROUTINE *****
SUB READ.AD (ADCHAN, VAD!)

```

```

  WAIT STAT.REG, COM.WAIT
  OUT COM.REG, CADIN
  WAIT STAT.REG, WRITE.WAIT, WRITE.WAIT
  OUT DATA.REG, 0
  WAIT STAT.REG, WRITE.WAIT, WRITE.WAIT
  OUT DATA.REG, ADCHAN

```



```

WAIT STAT.REG, READ.WAIT
LOW = INP(DATA.REG)
WAIT STAT.REG, READ.WAIT
HIGH = INP(DATA.REG)
DATA.VALUE! = HIGH * 256 + LOW
IF DATA.VALUE! > 32767 THEN DATA.VALUE! = DATA.VALUE! - 65536!
VAD! = RES! * DATA.VALUE! - OFFSET

```

END SUB

```

REM ***** SUBROUTINE TO PROGRAM THE DMA CONTROLLER CHIP *****
SUB SET.DMA.CONTROLLER (DMAMODE, NC)

```

```

DMACOUNT# = CDBL((NC * 2) - 1)
DMACOUNTH = INT(DMACOUNT# / 256)
DMACOUNTL = DMACOUNT# - DMACOUNTH * 256

```

```

OUT MODEREG, DMAMODE      ' Set DMA mode.
OUT BYTEREG, DMACHAN      ' Clear byte flip-flop.
OUT BASEREG, DMABASEL    ' Set lo byte DMA memory base address.
OUT BASEREG, DMABASEH    ' Set hi byte DMA memory base address.
OUT COUNTREG, DMACOUNTL  ' Set lo byte DMA byte count.
OUT COUNTREG, DMACOUNTH  ' Set hi byte DMA byte count.
OUT PAGEREG, DMAPAGE     ' Set DMA memory page.
OUT MASKREG, DMACHAN     ' Clear DMA channel mask.

```

END SUB

```

REM ***** SUBROUTINE TO SET UP DT-2818 FOR DMA *****
SUB SET.DMA.DT2818 (TICKS, START.CHAN, END.CHAN)

```

REM Set-up DT-2818 for continuous DMA operation.

```

REM Write SET CLOCK PERIOD command.
  WAIT STAT.REG, WRITE.WAIT, WRITE.WAIT
  WAIT STAT.REG, COM.WAIT
  OUT COM.REG, CCLOCK
REM Write high and low bytes of TICKS.
  TICKSH = INT(TICKS / 256)
  TICKSL = TICKS - TICKSH * 256
  WAIT STAT.REG, WRITE.WAIT, WRITE.WAIT
  OUT DATA.REG, TICKSL
  WAIT STAT.REG, WRITE.WAIT, WRITE.WAIT
  OUT DATA.REG, TICKSH
REM Write SET A/D PARAMETERS command.
  WAIT STAT.REG, WRITE.WAIT, WRITE.WAIT
  WAIT STAT.REG, COM.WAIT
  OUT COM.REG, CSAD
REM Write A/D gain byte. (Always 0 for DT-2818)
  WAIT STAT.REG, WRITE.WAIT, WRITE.WAIT

```

```

    OUT DATA.REG, 0
    REM Write A/D start channel byte.
    WAIT STAT.REG, WRITE.WAIT, WRITE.WAIT
    OUT DATA.REG, START.CHAN
    REM Write A/D end channel byte.
    WAIT STAT.REG, WRITE.WAIT, WRITE.WAIT
    OUT DATA.REG, END.CHAN
    REM Write two bytes, dummy number of conversions.
    WAIT STAT.REG, WRITE.WAIT, WRITE.WAIT
    OUT DATA.REG, &H5
    WAIT STAT.REG, WRITE.WAIT, WRITE.WAIT
    OUT DATA.REG, &H5

```

END SUB

```

    REM ***** SUBROUTINE TO SMOOTH DATA *****
    SUB SMOOTH (M, Z, d!(), N)

```

```

    REM Data is smoothed "M" times by "Z" point approximation
    LIM1 = INT(Z / 2 + 1): LIM2 = LIM1 - 1
    FOR J = 1 TO M
        FOR I = LIM1 TO N - LIM2
            DAT1! = 0
            FOR K = -LIM2 TO LIM2
                DAT1! = DAT1! + d!(I + K) / Z
            NEXT
            d!(I) = DAT1!
        NEXT
    NEXT

```

END SUB

```

    REM ***** SUBROUTINE TO STOP AND CLEAR THE DT-2818 *****
    SUB STOP.AND.CLEAR

```

```

    OUT COM.REG, CSTOP
    TEMP = INP(DATA.REG)
    WAIT STAT.REG, WRITE.WAIT, WRITE.WAIT
    WAIT STAT.REG, COM.WAIT
    OUT COM.REG, CCLEAR
    CALL CHECK.ERROR

```

END SUB

```

    REM ***** WRITE BINARY DATA FILE SUBROUTINE *****
    SUB WRITE.BINARY (Filename$, DAT!(), TOTAL&, TIMESTEP!)

```

```

    PRINT "Writing "; Filename$; " ..."
    OPEN Filename$ FOR BINARY ACCESS WRITE AS #1

```

```
    PUT #1, , TOTAL&: PUT #1, , TIMESTEP!  
    FOR I = 1 TO TOTAL&  
        PUT #1, , DAT!(I)  
    NEXT  
CLOSE #1
```

```
END SUB
```

```
REM ***** WRITE D-A IMMEDIATE SUBROUTINE *****  
SUB WRITE.DA (DACHAN, VOLTS!)
```

```
    DATA.VALUE! = CINT((VOLTS! + OFFSET) * FACTOR / RANGE)  
    IF DATA.VALUE! > (FACTOR - 1) THEN DATA.VALUE! = FACTOR - 1  
    WAIT STAT.REG, COM.WAIT  
    OUT COM.REG, CDAOUT  
    WAIT STAT.REG, WRITE.WAIT, WRITE.WAIT  
    OUT DATA.REG, DACHAN  
    HIGH = INT(DATA.VALUE! / 256!)  
    LOW = DATA.VALUE! - HIGH * 256!  
    WAIT STAT.REG, WRITE.WAIT, WRITE.WAIT  
    OUT DATA.REG, LOW  
    WAIT STAT.REG, WRITE.WAIT, WRITE.WAIT  
    OUT DATA.REG, HIGH
```

```
END SUB
```

## REFERENCES

- Altintas, Y., Yellowley, I., Tlusty, J., "The Detection of Tool Breakage in Milling," Sensors and Controls for Manufacturing, PED-Vol. 18, Proceedings of the Winter Annual Meeting, Miami Beach, Florida, pp. 41-48, ASME, New York, 1985.
- Automation Intelligence, "FlexMate Motion Co-Processor Installation and Maintenance Manual," Automation Intelligence, Inc., Orlando, Florida, 1987.
- Automation Intelligence, "FlexMate Machining Center Operator's Guide", Automation Intelligence, Inc., Orlando, Florida, 1989.
- Balakrishnan, P., Kannatey-Asibu, E., Trabelsi, H., Emel, E., "A Sensor Fusion Approach to Cutting Tool Monitoring," Advances in Manufacturing Systems Integration and Processes, Proceedings of the 15th Conference on Production Research and Technology, Berkeley, California, pp. 101-108, SME, Dearborn, Michigan, 1989.
- Birla, S., "Report on Sensors for Adaptive Control and Machine Diagnostics," General Motors Technical Center, General Motors Corporation, Warren, Michigan, 1980.
- Bollinger, J.G., Duffie, N.A., Computer Control of Machines and Processes, Addison-Wesley, New York, 1988.
- Data Translation, "User Manual for DT-2801 Series Single Board Analog and Digital I/O Systems," Data Translation, Inc., Marlboro, Massachusetts, 1988.
- Delio, T.S., "A Sensor-Based Adaptive Control Constraint System for Automatic Spindle Speed Regulation to Obtain Highly Stable Milling," Ph.D. Dissertation, Mechanical Engineering Department, University of Florida, Gainesville, 1989.
- Delio T., Smith, S., Tlusty, J., Zamudio, C., "Stiffness, Stability, and Loss of Process Damping in High Speed Machining," Fundamental Issues in Machining, PED-Vol. 43, Proceedings of the Winter Annual Meeting, Dallas, Texas, pp. 171-191, ASME, New York, 1990.

- Eman, K., Wu, S.M., "A Feasibility Study of On-Line Identification of Chatter in Turning Operations," *Journal of Engineering for Industry*, ASME, Vol. 102, pp. 513-321, November, 1980.
- GenRad, "GR 2515 Computer-Aided Test System Operating Manual," GenRad, Inc., Santa Clara, California, 1985.
- Johnson, C.M., Richter, F., Spiewak, S.A., "A Comprehensive Model of the Milling Process for On-Line Tool Condition Monitoring," *Proceedings of the 16th NAMRC*, Urbana, Illinois, pp. 178-189, SME, Dearborn, Michigan, 1988.
- Keyvanmanesh, A.E., "Evaluation of Chatter Detection and Control System," *Master's Thesis*, Mechanical Engineering Department, University of Florida, Gainesville, 1990.
- Lan, M.-S., Naerheim, Y., "In-Process Detection of Tool Breakage in Milling," *Sensors and Controls for Manufacturing*, PED-Vol. 18, *Proceedings of the Winter Annual Meeting*, Miami Beach, Florida, pp. 49-56, ASME, New York, 1985.
- Lauderbaugh, L.K., Ulsoy, A.G., "Model Reference Adaptive Force Control in Milling," *Modeling, Sensing, and Control of Manufacturing Processes*, PED-Vol. 23, *Proceedings of the Winter Annual Meeting*, Anaheim, California, pp. 165-179, ASME, New York, 1986.
- Matsushima, K., Bertok, P., Sata, T., "In-Process Detection of Tool Breakage by Monitoring the Spindle Motor Current of a Machine Tool," *Measurement and Control for Batch Manufacturing*, *Proceedings of the Winter Annual Meeting*, Phoenix, Arizona, pp. 145-153, ASME, New York, 1982.
- Okafor, A.C., Marcus, M., Tipirneni, R., "Multiple Sensor Integration Via Neural Networks for Estimating Surface Roughness and Bore Tolerance in Circular End Milling," *Proceedings of the 18th NAMRC*, University Park, Pennsylvania, pp. 128-136, SME, Dearborn, Michigan, 1990.
- Press, W.H., Flannery, B.P., Teukolsky, S.A., Vetterling, W.T., Numerical Recipes, Cambridge University Press, New York, 1986.
- Pressman, R.S., Williams, J.E., Numerical Control and Computer Aided Manufacturing, John Wiley & Sons, New York, 1977.
- Principe, J.C., Yoon, T., "A Numeric-Symbolic Approach to Machine Tool Supervision," *Sensors Expo West Proceedings*, Long Beach, California, pp. 301B1-301B8, Helmers Publishing, Peterborough, New Hampshire, 1990.

- Richter, F., Spiewak, S.A., "A System for On-Line Detection and Prediction of Catastrophic Tool Failure in Milling," Proceedings of the 17th NAMRC, Columbus, Ohio, pp. 137-143, SME, Dearborn, Michigan, 1989.
- Smith, K.S., "Chatter, Forced Vibrations, and Accuracy in High-Speed Milling," Master's Thesis, Mechanical Engineering Department, University of Florida, Gainesville, 1985.
- Smith, K.S., "Automatic Selection of the Optimum Spindle Speed in High-Speed Milling," Ph.D. Dissertation, Mechanical Engineering Department, University of Florida, Gainesville, 1987.
- Smith, S., Delio, T., "Sensor-based Control for Chatter-free Milling by Spindle Speed Selection," Control Issues in Manufacturing Processes, DSC-Vol. 18, Proceedings of the Winter Annual Meeting, San Francisco, California, pp. 107-114, ASME, New York, 1989.
- Smith, S., Tlusty, J., "Update on High Speed Milling Dynamics," Journal of Engineering for Industry, ASME, Vol. 112, pp. 142-149, May, 1990.
- Tarnag, Y.-S., "Sensing of Tool Breakage in Milling," Master's Thesis, Mechanical Engineering Department, University of Florida, Gainesville, 1986.
- Tarnag, Y.-S., "Use of Various Signals for Milling Cutter Breakage Detection," Ph.D. Dissertation, Mechanical Engineering Department, University of Florida, Gainesville, 1988.
- Tlusty, J., "Machine Dynamics," Handbook of High Speed Machining Technology, R.I. King, ed., Chapman and Hall, New York, 1985.
- Tlusty, J., Andrews, G.C., "A Critical Review of Sensors for Unmanned Machining," CIRP Annals, Vol. 32-2, pp. 563-572, 1983.
- Tlusty, J., Smith, S., "High Speed High Power Milling," Advances in Manufacturing Systems Integration and Processes, Proceedings of the 15th Conference on Production Research and Technology, Berkeley, California, pp. 215-222, SME, Dearborn, Michigan, 1989.
- Tlusty, J., Tarnag, Y.-S., "Sensing Cutter Breakage in Milling," CIRP Annals, Vol. 37-1, pp. 45-51, 1988.

- Ulusty, J., Tyler, T., "Adaptive Control for Die Milling: Criteria and Strategies," Computer-Aided Design and Manufacture of Dies and Molds, PED-Vol. 32, Proceedings of the Winter Annual Meeting, Chicago, Illinois, pp. 45-60, ASME, New York, 1988.
- Tyler, T.R., "Adaptive Control for Preventing Breakage of Flexible End Mills," Master's Thesis, Mechanical Engineering Department, University of Florida, 1989.
- Ulsoy, A.G., Koren, Y., Rasmussen, F., "Principal Developments in the Adaptive Control of Machine Tools," Measurement and Control for Batch Manufacturing, Proceedings of the Winter Annual Meeting, Phoenix, Arizona, pp. 105-119, ASME, New York, 1982.
- Vierck, C., "MTL Progress Report: Cutter Breakage," Internal Report, Machine Tool Laboratory, Mechanical Engineering Department, University of Florida, Gainesville, 1991.
- Walters, R. "A Distributed Implementation for In-Process Monitoring of Machine Tools," Ph.D. Dissertation, Electrical Engineering Department, University of Florida, in progress, 1991.
- Weck, M. Verhaag, E., Gather, M., "Adaptive Control for Face Milling Operations with Strategies for Avoiding Chatter Vibrations and for Automatic Cut Distribution," CIRP Annals, Vol. 24-1, pp. 405-409, 1975.
- Wells, R.L., "Maintaining the Supervision System Interface," Internal Report, Machine Tool Laboratory, Mechanical Engineering Department, University of Florida, Gainesville, 1991a.
- Wells, R.L., "DMA Data Acquisition Using the DT-2818 A/D-D/A Board and its Application in the Program PCDATA," Internal Report, Machine Tool Laboratory, Mechanical Engineering Department, University of Florida, Gainesville, 1991b.
- Wells, R.L., "Modifications to the Adaptive Control System," Internal Report, Machine Tool Laboratory, Mechanical Engineering Department, University of Florida, Gainesville, 1991c.
- White-Sunstrand, "Electrical Maintenance," White-Sunstrand Machine Tool Company, Belvidere, Illinois, 1983a.
- White-Sunstrand, "Mechanical Maintenance," White-Sunstrand Machine Tool Company, Belvidere, Illinois, 1983b.

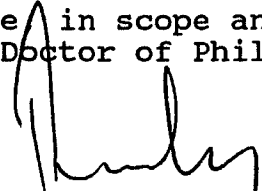
- Wright, P., Greenfield, I., Hayes, C., "A Prototype of a 'Next Generation-Control' Environment: Expert Systems for Planning and Sensor Integration," Proceedings of the 18th NAMRC, University Park, Pennsylvania, pp. 322-328, SME, Dearborn, Michigan, 1990a.
- Wright, P.K., Hansen, F.B., Pavlakos, E., "Tool Wear and Failure Monitoring on an Open Architecture Machine Tool," Fundamental Issues in Machining, PED-Vol. 43, Proceedings of the Winter Annual Meeting, Dallas, Texas, pp. 211-228, ASME, New York, 1990b.
- Yoon, T., "A Numeric/Symbolic Approach to Machine Tool Supervision," Ph.D. Dissertation, Electrical Engineering Department, University of Florida, Gainesville, 1990.



## BIOGRAPHICAL SKETCH

After having worked for several years in the contract engineering industry as a mechanical designer, the author obtained his Bachelor of Science degree in mechanical engineering from the University of South Florida at Tampa in 1985. He returned to contract engineering, and in 1987 entered the University of Florida to further his education. He earned a Master of Science degree in mechanical engineering in 1988. Upon completion of his doctorate, the author plans to join a university faculty and pursue a career in teaching and research.

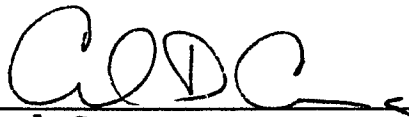
I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



---

Jiri Flusty, Chairman  
Graduate Research Professor of  
Mechanical Engineering

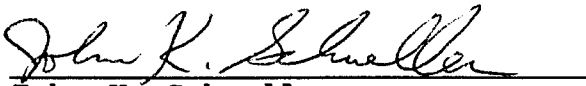
I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



---

Carl Crane  
Assistant Professor of  
Mechanical Engineering

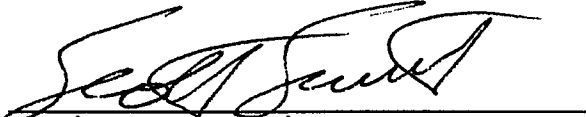
I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



---

John K. Schueller  
Associate Professor of  
Mechanical Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



---

Kevin Scott Smith  
Assistant Professor of  
Mechanical Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

*Senser Yeralan*

---

Senser Yeralan  
Associate Professor of  
Industrial and  
Systems Engineering

This dissertation was submitted to the Graduate Faculty of the College of Engineering and to the Graduate School and was accepted as partial fulfillment of the requirements for the degree of Doctor of Philosophy.

August, 1991

*Winfred M. Phillips*

---

*Dr.* Winfred M. Phillips  
Dean, College of Engineering

*Madelyn M. Lockhart*

---

Madelyn M. Lockhart  
Dean, Graduate School