THE TURNING WORKSTATION IN THE AMRF

NBSIR 88-3749

April 20, 1988

By:
A. Donmez
R. Gavin
L. Greenspan
K. Lee
V. Lee
J. Peris
E. Reisenauer
C. Shoemaker
C. Yang

U.S. DEPARTMENT OF COMMERCE National Bureau of Standards Gaithersburg, Maryland

THE TURNING WORKSTATION IN THE AMRF

Alkan M. Donmez
Robert J. Gavin
Lew Greenspan
Kang B. Lee
Vincent J. Lee
James P. Peris
Eric J. Reisenauer
Charles O. Shoemaker
Charles W. Yang

TABLE OF CONTENTS Page

LIST OF FIGURES

I. INTRODUCTION

This chapter gives a brief description of the Turning Workstation (TWS) developed at the National Bureau of Standards (NBS) and describes how this manual is organized. Finally, the chapter describes who should read this manual.

1. WHAT IS THE TWS?

The Turning Workstation is an automated turning center tended by an industrial robot for flexible manufacturing. The TWS was developed in the Automated Manufacturing Research Facility (AMRF) at the National Bureau of Standards (NBS) for the study of interface standards and to demonstrate the feasibility of the untended production of metal parts.

The TWS consists of a workstation controller, a high-level machine tool controller, a precision computerized numeric control (CNC) turning center served by a six-axis robot, a buffer turntable for local storage of tools, collets and robot gripper fingers, a tool-setting station, and a tray loading/unloading station for part blanks and finished parts.

In order to perform high-precision turning, collets are used for holding workpieces in the turning center. Part of the uniqueness of the TWS is that it facilitates standard-collet loading, collet changing, tool changing, and robot gripper-fingers changing. A five-axis micromanipulator, mounted on the wrist of the robot, is used to enhance the capability of the robot for loading the workpiece into the collet.

The TWS is designed to run in either stand-alone mode or integrated mode. In the stand-alone mode, part order information is entered into the console terminal. In the integrated mode, the workstation controller communicates with the higher level cell controller and accesses data information from the database through the AMRF network.

2. ABOUT THIS MANUAL

This section introduces the organization of the manual and the chapters contained therein.

2.1 How This Manual Is Organized

This manual is organized into twelve chapters, one appendix, a glossary, an index, and a list of references. This chapter, "Introduction", introduces the TWS and explains how to use this manual.

Chapter II, "Overview of the TWS", provides a brief description of the architecture of the TWS and how all the special systems of the TWS are designed to interact with one another.

Chapter III, "TWS Controller", is a detailed examination of the controller of the TWS, which orchestrates all activity within the TWS as well as between the TWS and higher levels of control.

Chapters IV and V, "High-Level Machine Tool Controller" and "Robot Controller Interface", respectively, provide information on the systems within the TWS that provide specific control for specific functions.

Chapters VI thru XII comprise the special systems of the TWS. As such, they provide more specialized information on innovations achieved in the AMRF at the National Bureau of Standards. These chapters are entitled "Robot Gripper", "Programmable Stop", "Micromanipulator", "Malfunction Detector", "Tool Setting Station", "Collet Changer", and "Turntable".

Appendix A contains the specific protocol for communication between the TWS controller and the specific devices within the TWS, while a glossary and index are provided for ease of reference.

3. WHO SHOULD USE THIS MANUAL

This document is intended for NBS personnel as well as government, industry, and university researchers interested in automated turning workstations as implemented in the AMRF. The manual assumes the reader is conducting research and developing applications for real-time control of robots in conjunction with turning workstations.

II. AN OVERVIEW OF THE TURNING WORKSTATION

The Automated Manufacturing Research Facility (AMRF) is situated at the National Bureau of Standards (NBS) [1,2,3]. It is an ongoing research effort in factory automation standards at NBS, in collaboration with the Manufacturing Technology Program of the U.S. Navy, industry, the academic community, and the Center for Manufacturing Engineering of NBS. The AMRF consists of six workstations: a Turning Workstation (TWS), a Horizontal Workstation, a Vertical Workstation, a Cleaning and Deburring Workstation, an Inspection Workstation, and a Material Handling Workstation. These workstations are integrated into a manufacturing cell and the workstation activities are scheduled and coordinated by a cell controller. The two higher-level controls, the facility and the shop controls, provide process planning, CAD design and engineering services, and offline programming of machine tools and robots. Data, commands, and status information are transmitted over a network communication system using a distributed data administration system.

The Turning Workstation is one of the more technologically advanced and flexible manufacturing workstations in the AMRF. As such, it addresses some of the problems associated with untended turning operations. These include automatic tool changing and workpiece loading. Some research has been done in automatic tool changing using the block tool system [4]. Other work has been accomplished in workpiece loading into a chuck using an adaptive force sensing technique [5]. Both of these systems use a specially designed and dedicated robot arm and gripper to perform the operations. The TWS takes another approach: it uses a general purpose commercial robot to perform robotic collet loading, robotic collet changing, and robotic tool changing in an automated environment. Tool changing is accomplished using a ball-lock tooling system [6]. Tool setting is performed by a touch calibration at the NBS-built tool setting station [7] when a tool is changed.

1. DESIGN GOALS FOR THE TURNING WORKSTATION

The Turning Workstation was developed with several design goals in mind. The goals are to develop a flexible and generic workstation that can be integrated into an automated factory, to improve the quality of parts produced, and to demonstrate the cost effectiveness of small batch manufacturing workstations.

1.1 Flexibility

Small batch factories characteristically produce a wide variety of parts in low volume. Most automated manufacturing workstations in existence today are not well suited for this type of environment. Many of them are programmed to execute a sequence of steps repeatedly, and any modification in the sequence needed in order to machine different parts requires extensive reprogramming. This is undesirable, because it reduces production time. Clearly, in order for an automated workstation to operate effectively in this environment, it must be designed with a great deal of flexibility. The workstation must be able to machine a variety of parts without requiring major modification. The Turning Workstation was designed with this flexibility in mind. Flexible features include the ability to machine a variety of parts without any changes to the workstation hardware or software. In order to achieve this, a myriad of engineering problems had to be investigated and solved. The following are some of the problems and their solutions:

1. Since different parts require different tools to machine them, tool changing must be automated in order to machine a variety of different parts without human intervention. The Turning Workstation robot is programmed to achieve this.

2. Different parts require different fixtures to hold them in the spindle of the turning center. Therefore, collet (the fixture used to hold the part) changing must be automated. Again, the Turning Workstation robot is programmed to achieve this.

3. The turning lathe requires different Numeric Control (NC) programs to machine different parts. The Turning Workstation Machine Tool Controller is designed with the ability of downloading various NC programs to the turning lathe.

1.2 Integrability

In order to demonstrate the feasibility of completely automating a small batch factory, it is essential for the TWS to perform as an integral part of this factory. The TWS is designed to operate in either integrated mode or stand-alone mode. In integrated mode, the TWS obtains batch orders and process information from the AMRF cell controller and the AMRF database. In stand-alone mode, the TWS obtains batch orders from a local operator, and process information from a local database.

1.3 Quality

As with any manufacturing operation, the quality of the product was a major concern in the development of the Turning Workstation. Several steps were taken to improve the quality of the parts made. One of these was to use collets to hold a part in place during machining. A collet fits very closely around the part it holds, thus allowing a higher degree of machining accuracy than that of conventional chucks. Another quality improvement feature was the addition of a high-level machine tool controller developed by NBS capable of real-time compensation of machine errors, including geometric and thermal errors.

1.4 Cost-Effectiveness

Several steps were taken to demonstrate the cost effectiveness of developing small batch manufacturing workstations. Most major components within the Turning Workstation are off-the-shelf items. For instance, the lathe is a Hardinge Brothers product and the robot used is a Bendix product. The use of a single robot manipulator in conjunction with a number of interchangeable gripper fingers also increased the flexibility of the robot without incurring a high cost.

2. WORKSTATION HARDWARE CONFIGURATION

The major components of the TWS include a turning lathe that machines parts, a robot that loads the lathe with parts, a buffer tray that stores the parts, and a buffer turntable that stores the tools and the fixtures needed by the lathe to machine parts. The workstation also contains other sophisticated devices and sensors that facilitate automated manufacture. Sensory information processed by the device controllers includes positional, thermal, force and torque, and vibrational data. For instance, the Machine Tool Controller obtains thermal and positional information from thermal couples and laser interferometers to perform error correction. Sensors used in the TWS include LVDT's (Linear Variable Differential Transducer), positional encoders, inductive proximity sensors, accelerometers, thermocouples, and laser interferometers.

A more detailed overview of the Turning Workstation is shown in Figure II.1. The TWS consists of a six degree-of-freedom, electrically-driven robot mounted on a gantry, a high-precision turning center with 0.25-micrometer (10-microinch) resolution, a microcomputer-based workstation controller, a high-level machine tool controller, and an array of sophisticated devices and sensors. These devices include a micromanipulator that acts as a "smart" wrist for the robot, a robot gripper system with two back-to-back grippers having interchangeable fingers, a collet-changing device, an "intelligent" servo-controlled local buffer turntable, a programmable stop for part length referencing, a programmable malfunction detector for monitoring the machining operation of the turning center, a tray station controller for coordinating tray-loading and tray-unloading operations with the Material Handling Station. The TWS system block diagram is shown in Figure II.2.

3. WORKSTATION CONTROLLER

The Turning Workstation controller is based on a Multibus computer, Intel System 310/286 , which uses an 80286 16-bit microprocessor as the CPU. The computer has 1 Mbyte of RAM, a floppy disk drive, a 40 Mbyte hard disk drive, and ten RS232C serial ports. The primary function of the TWS controller is to coordinate the device executions within the workstation, and it does this by communicating with the device controllers through the RS232C serial links. The TWS controller is data driven, and it obtains manufacturing data from either

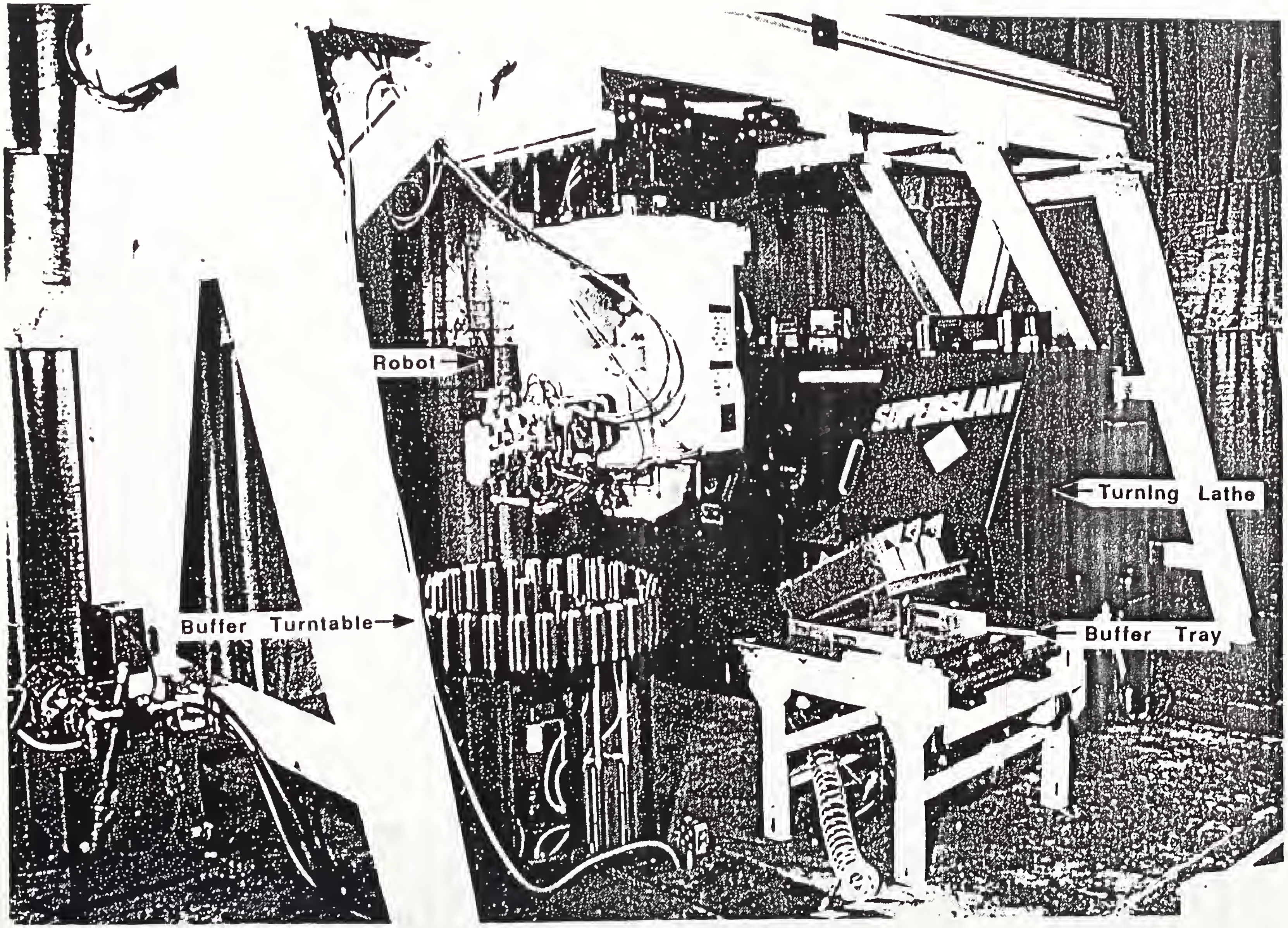


FIGURE II.1. THE OVERVIEW OF THE TURNING WORKSTATION

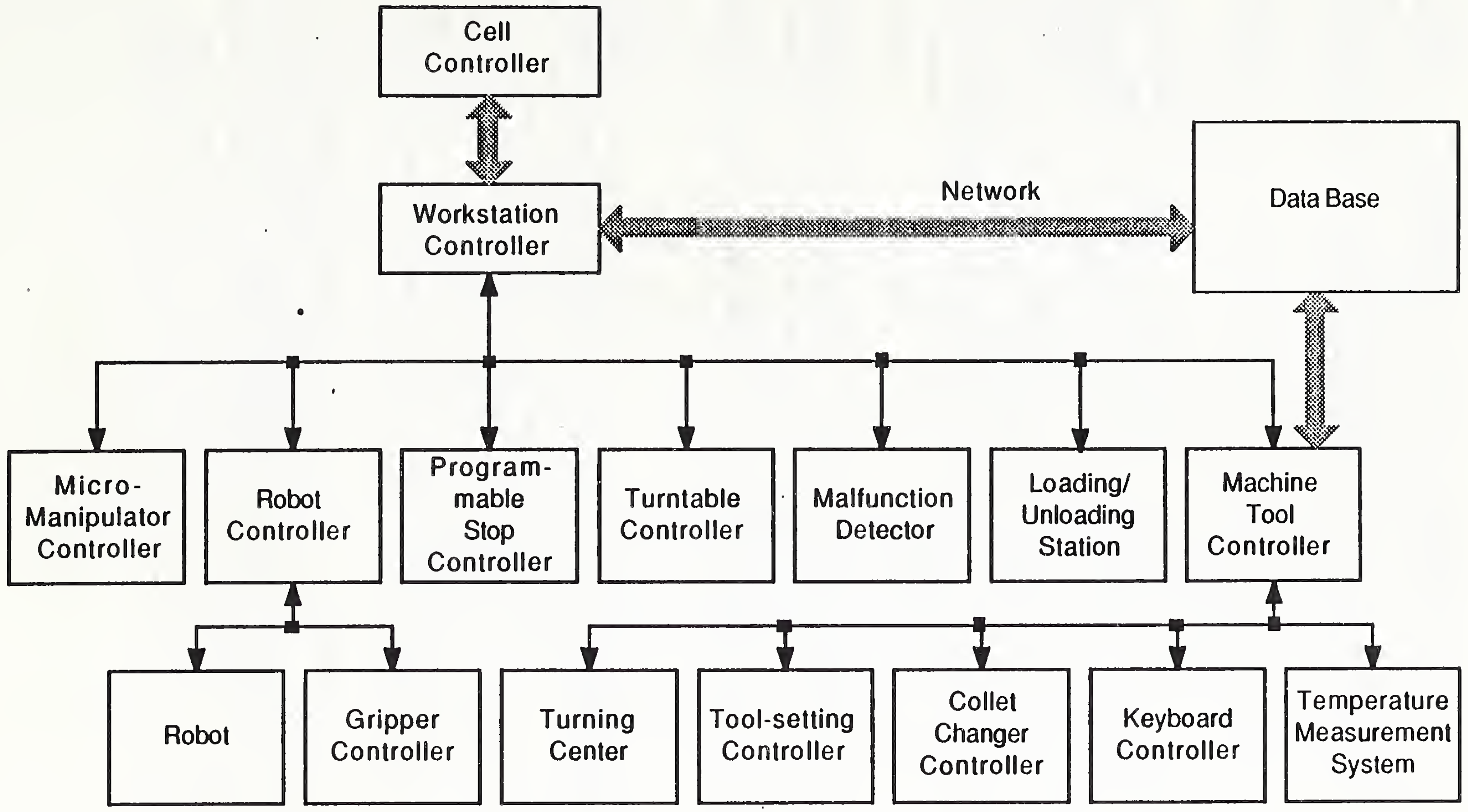


FIGURE II.2. THE TURNING WORKSTATION (TWS) SYSTEM BLOCK DIAGRAM

the AMRF database or a local database. The TWS controller software is an application using the iRMX 86. The iRMX 86 Operating System is used because it supports real-time applications and provides multitasking, which significantly aids the system design process. Also, multitasking simplifies the process of building an application system that processes real-time events. Thus the system code is less complex and easier to maintain.

The ability to react to asynchronous external events as they occur (in real-time) is an important characteristic of the TWS controller. A real-time algorithm is essential for effectively controlling the devices within a manufacturing workstation. This is because the status feedbacks from the devices occur asynchronously. To illustrate the need for a real-time algorithm, suppose a sequential algorithm is used to control the workstation. A situation where many devices contend for the attention of the controller can easily arise. If the controller is waiting on a transaction with a particular device while other devices are trying to update the controller with more vital information, the other devices would be ignored by the controller until the device in question has completed its transaction. Clearly, this is not acceptable. A real-time algorithm would be able to react to these external events and process all the information at the right time.

The multitasking capability is another important characteristic that allows the TWS controller to perform effectively. The TWS controller employs one task per device; this establishes a one-to-one relationship between device and task which simplifies application coding. This implementation avoids using a single program to process multiple, asynchronous status feedback from many devices. The different tasks that make up the workstation controller are listed in Figure II.3.

The function to be performed by the TWS is thought of as a hierarchy composed of three levels. At the highest level, the TWS is told to perform a "job". MACHINE_LOT, for example, is the job of machining a batch of parts (see Figure II.4). At the next level, a job is broken down into "task level" work elements, which are subtasks that must be performed in order to do the job. GET_BLANK, for example, is a task level work element executed by TWS to fetch a part from the buffer tray (see Figure II.5). At the last level, each task level work element is broken down into "device level" work elements, which contain the actual machine commands sent to the devices inside the workstation.

The structure of the TWS controller follows the hierarchical nature of the function performed by the TWS. The job level controller resides at the highest level and controls the execution of jobs. The task level controller resides at the next highest level and controls the execution of task level work elements. The device level controller resides at the lowest level and controls the execution of device level work elements. The job level controller obtains status feedback from the task level controller. The task level controller

TASK	MODULE	FUNCTION
CM_TASK	CO_MANAGER.P8	Console manager
LOC_SC_TASK	L0C_MON.P86	Local monitor manager
DVT_TASK	DVT_M0D.P86	Device test manager
DVT_STA	DVT_STA.P86	Allows device testing to be interrupted
DSP_TASK	SET_DISP.P86	Sets up various screen displays
SHELL_MOD	SHELLMOD.P86	Dummy task used for debugging
L1_TASK	TASK1.P86	Manages work element execution
L2_TASk	TASK2B.P86	Manages equipment element execution
	INIT_TASKB.P86	Creates the tasks
STA_CHG_TASK	STACHG.086	View or modify various device mailboxes
MM_TASK	MM.P86	Processes status from micromanipulator
MC_TASK	MC.P86	Processes status from machine tool controller
PS_TASK	PS.P86	Processes status from programmable stop
TT_TASK	TT.P86	Processes status from turntable
RT_TASK	RT.P86	Processes status from robot
GR_TASK	GR.P86	Processes status from gripper controller
DS_TASK	DS.P86	Processes status from malfunction sensor
TR_TASK	TR.P86	Processes status from tray controller

FIGURE II.3. LISTING OF TASKS THAT COMPRISE THE WORKSTATION CONTROLLER

```
SETUP_AREA

RECEIVE_TRAY

MACHINE_LOT

SHIP_TRAY

TAKEDOWN_AREA
```

FIGURE II.4. A LIST OF TYPICAL JOB ELEMENTS

```
//OPSHEET

0100        GET_BLANK

0201        LOAD_BLANK

0302        MACHINE1

0403        ROTATE_LD

0504        MACHINE2

0605        REMOVE

0706        UNLOAD_PT

//END_OPSHEET
```

FIGURE II.5. A TYPICAL OPERATION SHEET FOR THE WORKSTATION CONTROLLER

obtains status feedback from the device level controller and the device level controller receives status feedback from the devices.

The data used to drive the TWS controller is organized in a hierarchical manner, much like the controller architecture. Job data is used by the job level controller, task level work element data is used by the task level controller, and device level work element data is used by the device level controller. Job data contains information about the location of the task level work element data, and task level work element data contains information about the location of the device level work element data. Job data include parameters such as part-ID, number of parts to be made, and keys needed to find the process plan flat file that is associated with the job. The process plan flat file contains information such as tools and fixtures needed to machine the part, and the sequence of task level work elements that is to be executed to produce the part. This sequence is decomposed by the TWS controller into a task level work element sheet that is used by the task level controller. Each task level work element points to a device level work element sheet, which is executed by the device level controller. The sequence in which the task and device level work elements' instructions are executed is determined by its assigned step numbers and precedence steps. A work element cannot be executed until its precedence step requirement is fulfilled. In this manner, multiple work elements and equipment instructions can be carried out concurrently. Thus the workstation controller can run the workstation by following the operation sheet and executing equipment instructions step by step.

The workstation controller can operate in either stand-alone mode or integrated mode. In the stand-alone mode the workstation controller accepts part order information from a console terminal. It accesses data, such as process plans, work elements, equipment instructions, tool table, or tray configuration, etc. from the local data stored in the hard disk. Then it executes the process plan to accomplish the goal. In the integrated mode the workstation controller receives job commands from the cell controller through the AMRF network. It then communicates with the database manager via the network service to retrieve the proper process plan and data for the job. It can execute multiple jobs, or multiple work elements, or multiple equipment instructions concurrently if there is no interference between the different devices.

The workstation controller software is written in Programming Language for Microcomputers (PLM), a high-level structured language. The controller software developed is modular and designed for the ease of expandability. This allows the addition of other devices or low-level controllers to the workstation.

4. HIGH-LEVEL MACHINE TOOL CONTROLLER

The high-level machine tool controller is one level below the workstation controller in the Turning Workstation control hierarchy. It receives commands from the workstation controller and decomposes the tasks assigned by the workstation controller into several subtasks. By sending appropriate commands corresponding to these subtasks to low-level controllers, the high-level machine tool controller coordinates the operations of the turning center and its auxiliary equipment such as the tool-setting station, the temperature measurement system, and the automatic collet changer. At the end of each task, it sends status information to the workstation controller. In order to accomplish the actual part-cutting operation, the high-level machine tool controller accesses the AMRF database. When needed, it downloads necessary NC part programs from the database to the computerized numerical control (CNC) controller of the turning center, and activates the NC program to make a part on the turning center. It also has the capability of uploading any NC program from the CNC controller to the database.

Another function of the machine tool controller is the enhancement of the turning center accuracy by real-time software error compensation for geometric and thermally induced machine errors [9,10,11]. The errors are predicted as a function of nominal machine axis position, direction of motion, and a machine tool temperature profile based on previous calibration measurements. During machining, these errors are converted to machine servo counts and injected into the following error registers of the CNC controller. In the error compensation mode, the high-level machine tool controller runs in synchronization with the CNC controller. Thus position information is updated and error correction is injected every servo control cycle of 20 milliseconds. With the error compensation system implemented, the overall machine accuracy is improved by a factor of 10 to 20.

The high-level machine tool controller is a Multibus single-board microcomputer with a 128 KByte RAM memory. It has a 16-bit 8086 microprocessor as CPU, and a 8087A numeric coprocessor for floating point arithmetic operations. In order to meet the high servo bandwidth requirement of performing error compensation during contouring cuts, the single-board microcomputer is run at an 8-MHz clock rate. The high-level machine tool controller uses two multibus serial input/output (I/O) boards and three multibus parallel I/O boards for interfacing to other devices. It communicates with the turning center CNC controller through parallel ports for error compensation purposes and uses a serial port for NC part program uploading and downloading. The communications with the database and the workstation controller are done through the AMRF network. However, the machine tool controller can operate in a stand-alone mode by communicating with the workstation controller through an RS232C serial interface. The communications between the machine tool controller and other lower level controllers are done via serial interfaces.

The software for the high-level machine tool controller is written in PLM. The software is written in a modular fashion for easy maintainability. In addition to integrated and stand-alone operating modes, four different modes are coded in the software for debugging purposes. These modes are: 1) communication test mode between machine controller and the workstation controller only, 2) communication test mode between machine controller, workstation controller, database, and turning center CNC, 3) local (serial) communication test mode between machine controller and the workstation controller, and 4) manual test mode to check the operations of the auxiliary equipment. In the manual test mode, commands to the high-level machine tool controller are entered through a CRT terminal. In all other modes, the machine tool controller functions as a slave of the workstation controller.

5. MICROMANIPULATOR

High precision machining is studied in the Turning Workstation. Hence, instead of the conventional, less accurate chucking device, a collet is chosen as the workpiece holding device. Due to a lack of clearance between the part and the collet, a mere 125 micrometers (0.005 inch), a robot with high positioning accuracy is required for collet loading operations. The positioning repeatability of the TWS robot is approximately 0.5 millimeter (0.020 inch) in steady-state conditions, and thermal drift during the warm-up period degrades this even further. In order to bring the robot from cold start to normal operating temperature, two to three hours of continuous exercise are required. During this robot warm-up period, valuable machining time is wasted. To eliminate the wasteful warm-up time and to make collet loading possible, a micromanipulator was designed at NBS [12] to assist the robot in performing collet loading, collet changing and tool changing operations.

The micromanipulator is a hydraulically driven, "intelligent", fine positioning device mounted between the wrist of the robot and the grippers. A picture of the micromanipulator and the robot grippers is shown in Figure II.6. The micromanipulator has five degrees of freedom, two rotational and three translational. All axes in the micromanipulator are servo controlled with a control cycle of 20 milliseconds to provide compliance. The micromanipulator has only position sensors, but disturbance of the servo control on a rotational axis provides an indirect touch sensation. In operation, the micromanipulator performs search and insertion routines to place a part into the collet. The search routine involves only translational movements, while the insertion routine needs all degrees of freedom in order to provide compliance for insertion. With inductive proximity probes providing position feedback, the positioning repeatability of the micromanipulator is better than 25 micrometers (0.001 inch). A picture of the gripper holding a part blank prior to insertion into the collet is shown in Figure II.7.

The micromanipulator is 6.35 centimeters (2.5 inches) thick and 12.7 centimeters (5 inches) square and weighs only 11 kilograms (5 lb). The micromanipulator controller is based on a single-board microcomputer, which has an 8086 microprocessor as CPU and an 8087A as arithmetic coprocessor for floating point calculations. It is interfaced to the workstation controller through an RS232C serial link operating at 9600 baud.

6. ROBOT GRIPPER

To efficiently handle multiple parts, cutting tools and collets, an "intelligent" robot gripper system with multiple and changeable fingers is needed. Because no existing commercial robot gripper met the functional

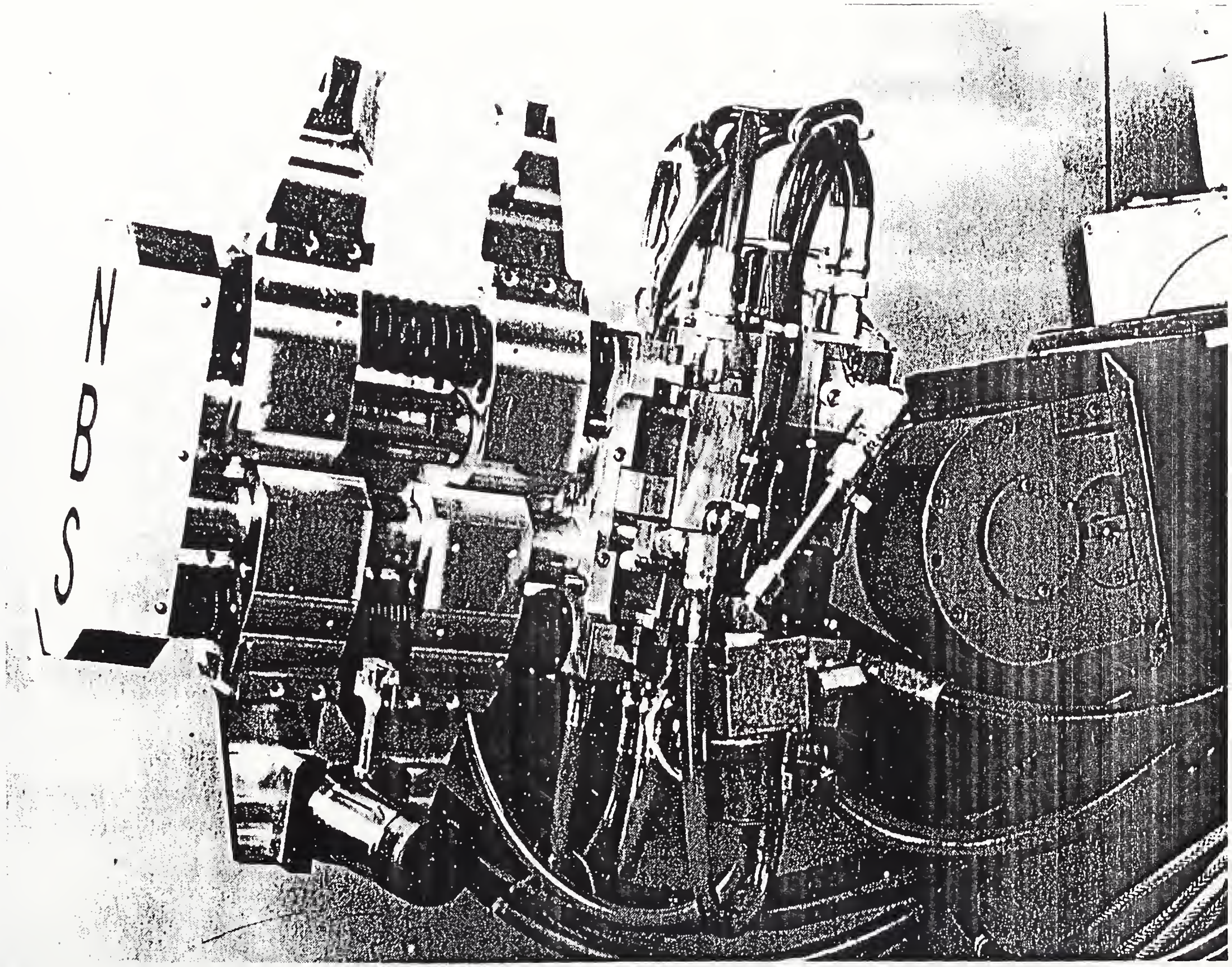


FIGURE II.6. THE ROBOT GRIPPERS AND THE MICROMANIPULATOR

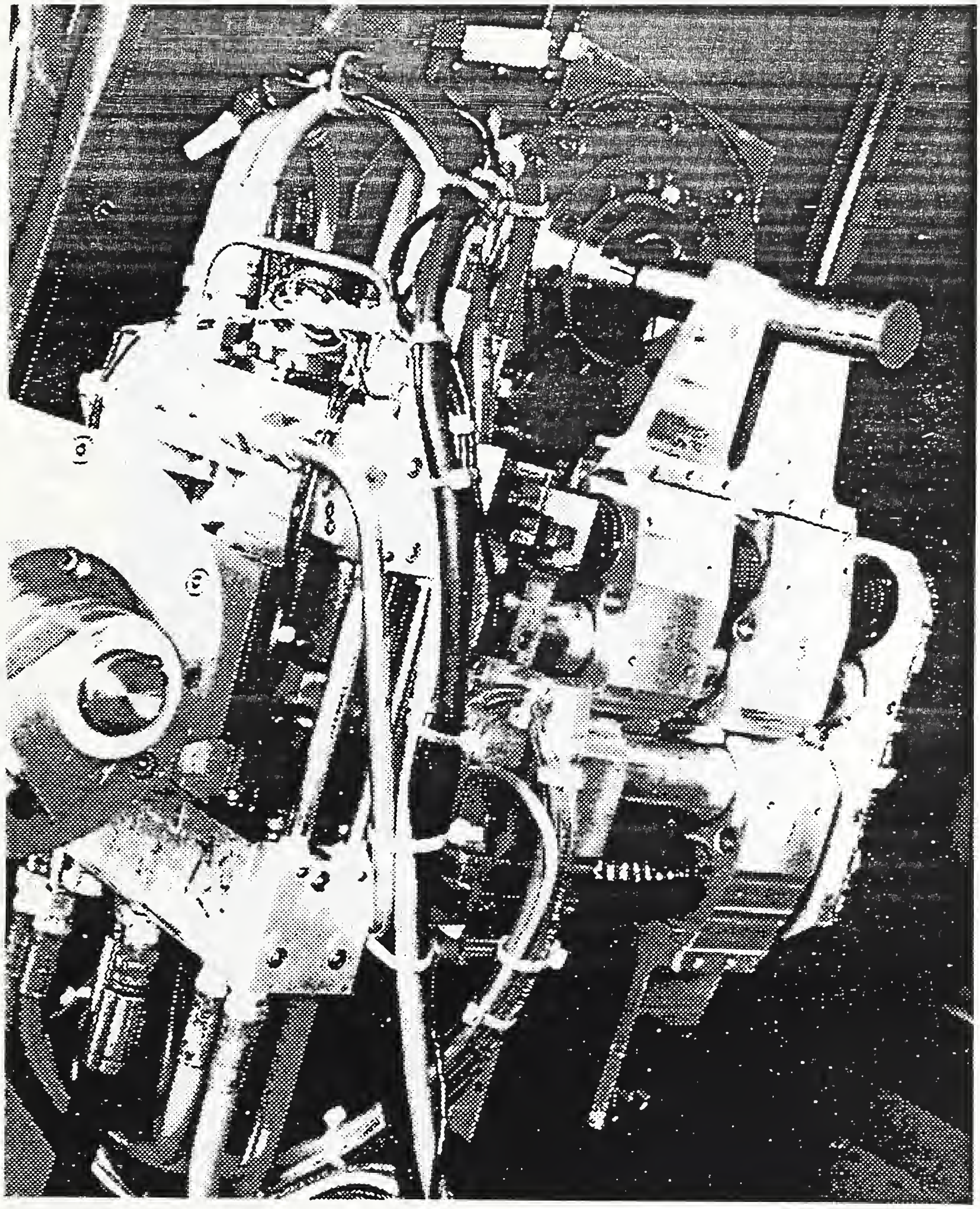


FIGURE II.7. GRIPPER HOLDING PART PRIOR
TO INSERTION INTO THE COLLET

requirements, a gripper system was designed at NBS to provide the necessary performance characteristics. This system has two grippers back-to-back with changeable fingers, which are held and aligned to the grippers by dovetail slots. Both grippers can be controlled independently and have the capability of closing the fingers to a specified position, ranging from 150 millimeters (5.9 inches) to within 2 millimeters (0.08 inch). In addition, each gripper is designed to have maximum strength and minimum weight. It has a maximum gripping force of 4,500 newtons (approx. 1000 lbf) and the entire end-effector weights only 9 kilograms (20 lb). The grippers are driven by air motors and are controlled by a single chip 8751 microcomputer. The gripper controller communicates to the host robot controller through an RS232C serial link operating at 1200 baud.

7. TURNTABLE

An "intelligent" local buffer storage unit is needed to store cutting tools, gripper fingers, collets, and parts in order to provide fast and efficient operation at the workstation. The turntable was designed at NBS to store up to sixty-six different items. It is 91.4 centimeters (36 inches) in diameter and has a capacity for 90.7 kilograms (200 lb) of tooling and fixtures. The sixty-six different store positions are arranged in three concentric rings. Three linear guides, one for each ring, are mounted beneath the table. Toolings and fixtures are individually seated in the holders. The holders are placed at each store position and are held straight up on the table by two guide pins. When the turntable indexes to a commanded position, a linear guide is actuated and extended through a hole on the table under the holder to push the holder up. Then the robot picks up the item presented and the linear guide retracts. These sequences of operations are coordinated by the workstation controller. A view of the gripper fingers changing operation is presented in Figure II.8. The turntable is servo-controlled by a motor controller using pulse-width modulation (PWM). The turntable controller is based on a 8751 single chip microcomputer which has 128 bytes of RAM and 4 Kbytes of ROM. It communicates with the workstation controller via RS232C protocols at 1200 baud.

8. COLLET CHANGER

In order for the robot to change collets, a microcomputer-controlled collet changing mechanism was developed and installed on the turning center. This is the first known use of a robotic system to change and to load standard collets on a turning center. The basic function of the changer is to loosen or to tighten the collet at the spindle and properly engage and disengage the clutch assembly so that the robot can change the collet. The controller designed for this device is highly sensor interactive. The changer is driven by a dc servomotor using PWM. The motor current is monitored by the controller to sense the right amount of torque applied on the drawbar in order to seat the collet in the spindle properly. The position of the drawbar, sensed by an inductive proximity sensor, is used as feedback for the servo controller to align the drawbar to the spindle. A photograph of the robot changing the

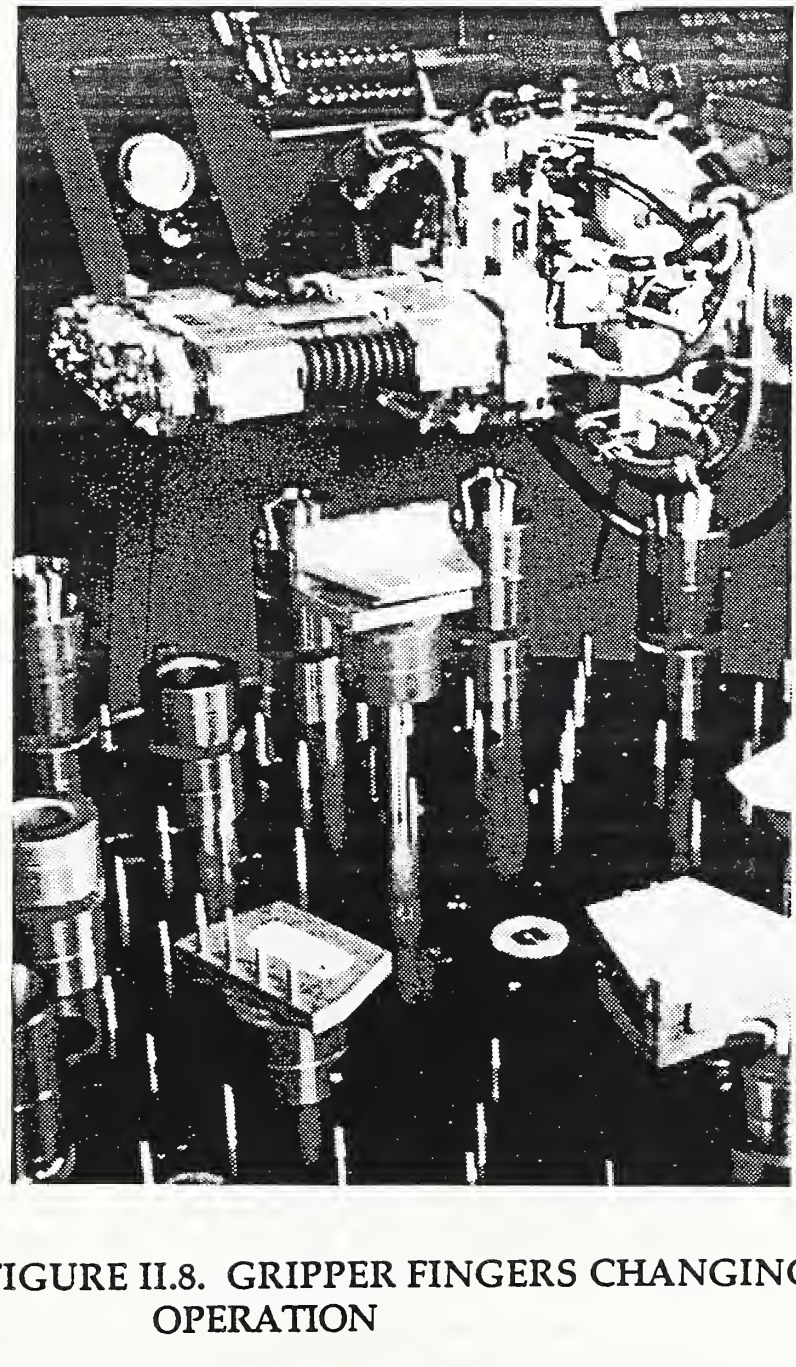


FIGURE II.8. GRIPPER FINGERS CHANGING OPERATION

collet is shown in Figure II.9. The collet changer controller, a single chip 8751 microcomputer-based system with 4 Kbytes of read only memory (ROM), is interfaced to the high-level machine tool controller via RS232C link operating at 1200 baud.

9. PROGRAMMABLE STOP

In order to use standard collets in the turning center, a part stop must be provided inside the collet. Designed at NBS, the programmable stop is a motor-driven leadscrew mechanism inside the spindle of the turning center. It is an accurate positioning device that can provide a variable reference length stop for the part in the collet. Driven by a servomotor and with a 11-bit absolute encoder as position feedback, the programmable stop produces a positioning repeatability of better than 2.5 micrometers (100 microinches) over a 17-centimeter (7-inch) travel. The stop mechanism and the controller were designed, built, and adapted to the lathe with a minimum of mechanical modification to the lathe. The 8751 microcomputer-based controller, which is similar to the turntable controller in design, communicates with the workstation controller through an RS232C interface at 1200 baud.

10. MALFUNCTION DETECTOR

One of the objectives of the TWS research is to study control and sensor interaction and problems associated with sustained workstation operation such as a 24-hour continuous operation. In order to do this, some kinds of sensors are required to detect abnormal operating conditions in the turning center. A malfunction detector, which monitors machining vibration, was developed at NBS for this purpose. The malfunction detector uses two accelerometers mounted on the two lathe tool turrets to measure vibration induced by machining. The accelerometer signals are summed and then root mean square (RMS)-detected with a 20-millisecond time constant. This conditioned signal is compared in real-time with previous calibration data. If the signal is smaller or greater than the stored data, the malfunction detector sends an appropriate warning signal and data to the workstation controller. The workstation controller analyzes the data and takes appropriate actions, such as to direct the robot to pick up a new blank if a part is dropped from the collet, or to halt the workstation operation if a tool is broken. Based on the signature of the vibration signal, the malfunction detector can detect the absence of a part to be machined, excessively dull tools, broken tools, an improper sequence of operations, and excessive time for any operation or between operations. The malfunction detector communicates with the workstation controller through an RS232C link at 9600 baud. With the malfunction detector, the TWS was able to perform a continuous untended operation for 25 hours producing more than 150 standard reference material (SRM) containers. In other untended test runs, more than 1,000 pieces of SRM containers were made in the TWS over a two-week period. During this period, several tool breakages due to machine tool failure were detected by the malfunction detector.

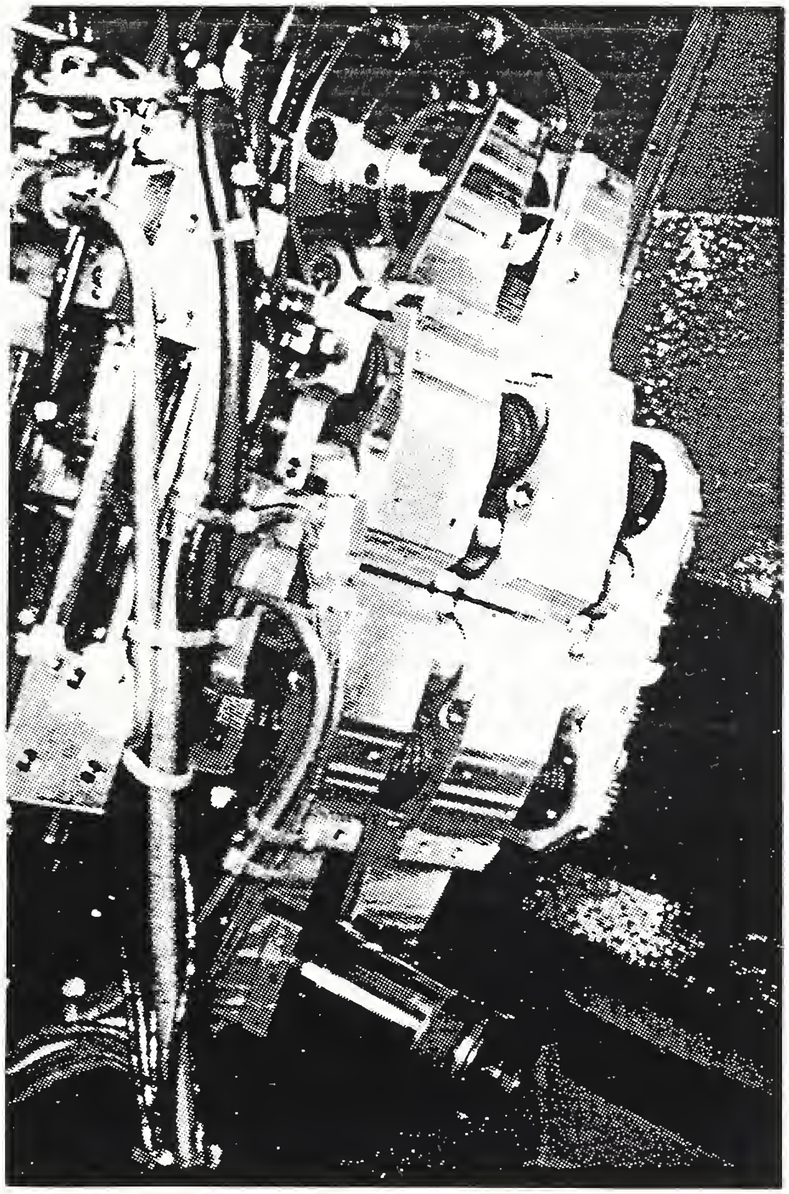


FIGURE II.9. COLLET CHANGING OPERATION BY ROBOT

11. NETWORK COMMUNICATIONS

The AMRF networking service provides communication connections between the different levels of controllers. It transfers control and data information on separate network paths via common memory mailboxes. Command and status information flow down and up the hierarchical control structure between supervisors and subordinates through mailboxes, while manufacturing data can be accessed in the database through other sets of mailboxes. There is no communication between controllers of the same control level. For instance, control command from one workstation controller to another workstation controller is prohibited. The AMRF network is a broadband token bus, using interfaces, gateways, and high-level protocols compatible with the General Motors Manufacturing Automation Protocols (MAP) and Technical Office Protocols (TOPS).

12. SUMMARY

An Automated Turning Workstation with the capabilities to perform robotic collet loading, collet changing, tool changing, and tool setting for untended high precision machining operations was developed in the AMRF at the NBS. The workstation can operate in either an integrated mode or a stand-alone mode. In the integrated mode, the workstation controller can communicate with the higher level cell controller and access manufacturing data such as process plans, NC program files, robot program files, etc., from the database through an integrated network. In stand-alone mode, the workstation controller can operate untended by following the process plans and accessing the manufacturing data in the local database. As configured, this workstation has achieved many hours of untended operation in making parts.

III. THE TURNING WORKSTATION CONTROLLER

The Turning Workstation Controller (TWSC) is a software application based on a Multibus computer. Its main function is to orchestrate the individual device executions within the workstation for the purpose of manufacturing parts. Another function of the controller is to act as a subordinate to the factory planning controller. This allows the Turning Workstation to be an integral part of the automated factory.

In order for the workstation controller to achieve its purpose, it must perform a variety of functions. One of them is information exchange. For instance, in order for the workstation controller to coordinate the devices within the workstation, manufacturing process information must be retrieved by the workstation controller (from either a remote factory database or a local workstation database) in order for it to control the sequence of operations within the workstation. Another function is device control, which involves communication with the devices and device status processing. A third function of the workstation controller is to provide a user interface so that an operator can control the workstation and determine the state of the operation. The user interface includes status displays, as well as diagnostic functions such as the individual testing of the devices in the workstation.

The workstation controller was developed on the Intel System 310/286 computer, which uses the 80286 16-bit microprocessor as CPU. The computer has 1 Mbyte of RAM, a floppy disk drive, a 40 Mbyte hard disk drive, and ten RS232C serial ports. The station controller software, written in the PLM 86 programming language, is an application using iRMX 86; a real-time, multitasking, interrupt-driven operating system. The operating system allows the station controller to process commands and status feedbacks from a number of other controllers asynchronously and in real-time: that is, as events occur. It also allows the station controller to keep track of a number of processes concurrently. The station controller can operate in either integrated or stand-alone mode. In integrated mode, it operates as a subordinate to the AMRF cell controller and accesses manufacturing data from the AMRF database. In stand-alone mode, it receives commands from an operator via a local terminal and accesses manufacturing data which reside in a local database.

1. DESIGN PHILOSOPHY

The design philosophy behind TWSC can be divided into three areas: flexible and generic manufacturing process, integrability, and real-time control.

1.1 Flexible and Generic High-Level Manufacturing Process

A major goal in the development of the workstation controller is to design completely flexible software that facilitates the manufacture of a variety of parts within the geometrical limits of the turning machine. Each individual part has a unique process associated with it due to its unique geometry. Therefore, the controller should be able to download various processes on-line. In other words, manufacturing different parts which requires different processes should not involve software changes in the workstation controller: a data change would be needed instead.

Another objective is to provide a high-level, generic format for the manufacturing process data so that the user can easily create these data sheets. A manufacturing process is the sequence of steps taken by the devices of the workstation in order to manufacture a part. The process is broken up into interchangeable generic tasks or "work elements" (such as GET_BLANK, LOAD_BLANK, etc.) with specific parameters that are geometry dependent. These high level work elements are then decomposed by the controller into device level commands which are then sent to the devices.

1.2 Integration

Another objective in the design of the controller is to enable the workstation to operate as a subordinate to higher level controllers as well as a stand-alone station which receives commands from an operator. This is necessary in order for the workstation to perform as an integral part of an automated factory. In order to achieve this, the workstation controller must be able to access global information needed for manufacture as well.

1.3 Real-Time Multitasking Control

In order for a workstation controller to be effective, it must be able to execute manufacturing commands and process status information from both lower level device controllers and a higher level controller in real-time: that is, process commands and react to status input from other controllers as events occur. The approach taken in the development of the workstation controller is to use a multitasking interrupt-driven operating system. In this way, individual tasks can be assigned to monitor other controllers and command and status processing can be done on an interrupt basis. The operating system also allows the processing of data from the cell or the database, and diagnostic testing by the operator concurrently with the execution of the manufacturing process.

2. CONTROLLER HARDWARE DESCRIPTION

The computer on which the workstation controller is developed is the INTEL SYSTEM 310. The Central Processing Unit (CPU) for the computer is the INTEL 80286 16-bit microprocessor. The SYSTEM 310 has 1 megabyte of Random Access Memory (RAM), as well as a 40 megabyte Quantum Winchester hard disk, a 5 1/4 inch 315 Kbyte floppy drive, and a network common memory board. The Winchester hard disk is used for the local database, and communication with the AMRF cell controller and the AMRF database is achieved through the network common memory board. For serial input/output (I/O), the SYSTEM 310 is configured with a iSBC

8274 mother board with two RS232C serial ports, and an iSBC 188/48 serial communication board with eight RS232C ports. The two serial ports of the iSBC 8274 board are connected to a workstation controller monitor (operator's console) and a system console. The serial ports on the iSBC 188/48 are connected to device controllers within the workstation and operate at various baud rates.

3. CONTROLLER DESIGN

The workstation controller is a software application developed on the iRMX86 Operating System. The controller is composed of a collection of tasks (programs that perform specific functions) whose order of execution is determined by a process plan with predefined priorities and precedences. The tasks are arranged in a hierarchical fashion to facilitate manufacturing operation using predefined process plans. Process plans and other data can be accessed locally from the Winchester hard disk, as well as the AMRF database. Remote communication, such as communication with the cell controller or the AMRF database, is done through the use of network common memory boards. Local communication to devices within the workstation is achieved through serial links. Figure III.1 illustrates the control hierarchy of the workstation.

3.1 Manufacturing Process Data

Manufacturing process data is used by the workstation controller to determine the sequence and content of operation for the devices within the workstation in order to make parts. The data used by the station controller is hierarchical in character. The process is decomposed into three levels: job level work elements, task level work elements, and device or equipment level work elements. The job work element is the highest level and the most generic: MACHINE_LOT and SETUP_AREA are typical job level work elements. A job is decomposed into a sequence of lower level, less generic tasks. For instance, in order to machine a part, the workstation must get the blank, load the blank into the lathe, machine it, and unload it. These tasks are the task level work elements: GET_BLANK, LOAD_BLANK are examples. Finally, each task is further decomposed by the station controller into a sequence of device work elements. A task may require a sequence of device commands. For instance, in order to get a blank (execute the GET_BLANK work element), a command must be sent to the robot controller to move to a buffer storage area, and upon arrival, the gripper must grip the part, etc. These device commands are the device level work elements.

The sequence of work elements is stored in work element sheets. For instance, a sequence of task level work elements is stored in a task level work element sheet, and a sequence of device level work elements is stored in a device level work element sheet. The work element sheets contain the actual steps to be executed, as well as sequence and precedence information to enable the workstation to do the right thing in the right moment.

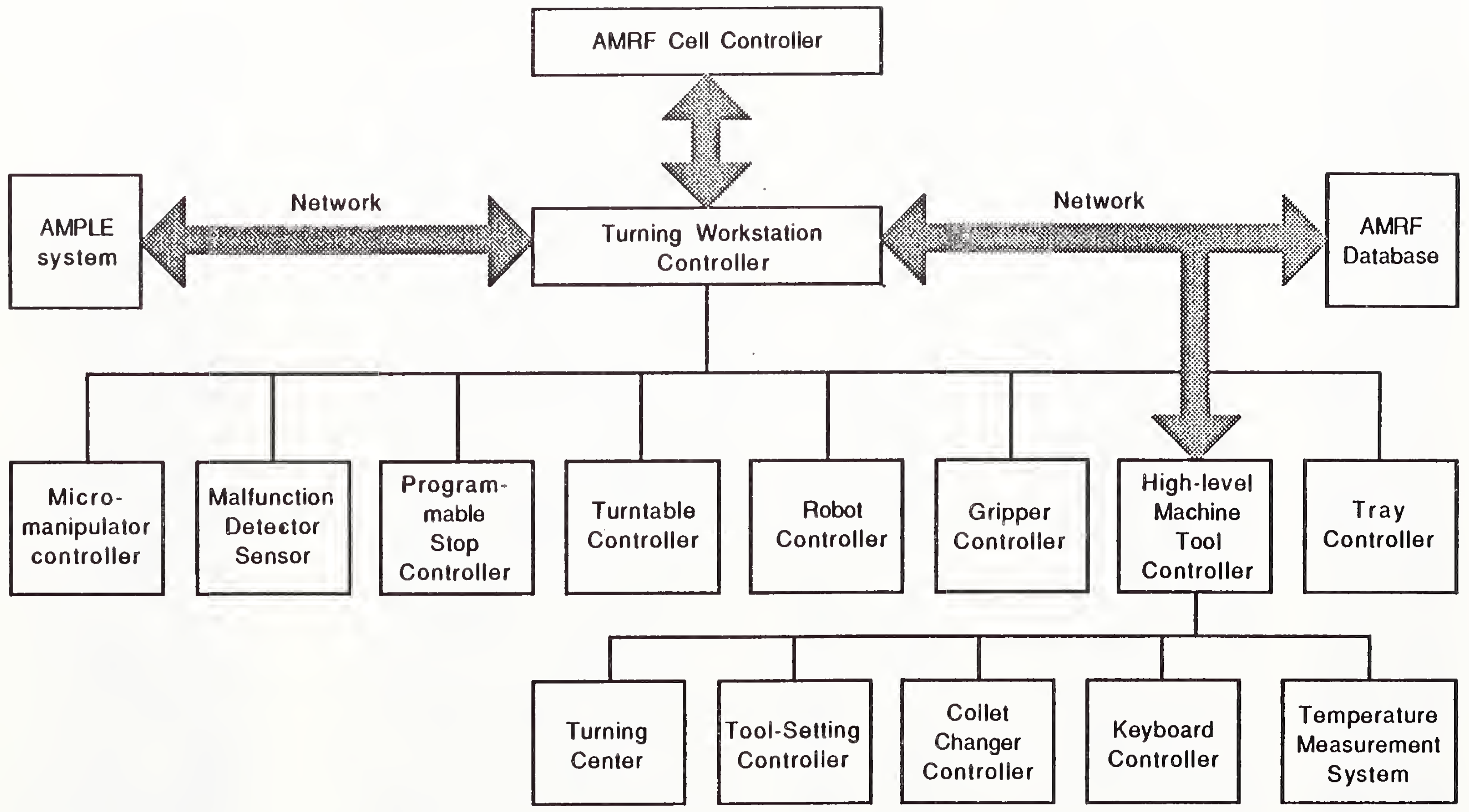

AMRF Cell Controller
AMPLE system
Network
Turning Workstation Controller
Network
AMRF Database
Micro-manipulator controller
Malfunction Detector Sensor
Program-mable Stop Controller
Turntable Controller
Robot Controller
Gripper Controller
High-level Machine Tool Controller
Tray Controller
Turning Center
Tool-Setting Controller
Collet Changer Controller
Keyboard Controller
Temperature Measurement System

FIGURE III.1. TURNING WORKSTATION CONTROL HIERARCHY

3.2 Controller Architecture

The station controller architecture is hierarchical (See Figure III.2). The manufacturing process control can be divided into three levels: task manager/job level , task level, and device level. The job level of the controller monitors the execution of jobs. Task execution is controlled and monitored at the task level while device operations are controlled at the device level. Below the device level are device monitors which communicate with device controllers and provide status feedback to be processed at the device level. The device level provides status feedback (the state of execution of a device) to the task level controller. Similarly, the task level provides the state of execution of a task level work element for the job level controller to process.

In addition to process control, various diagnostics tasks such as individual device tests or manufacturing process monitor display are controlled by the task manager.

The memory environment for the station controller is as follows:

- Network common memory is used for communication and data transfer between the station controller and the cell controller and the AMRF database. Communication with the cell and the database is achieved by using fixed-memory locations, or "mailboxes".

- iRMX 86 Operating System provides system calls which enable applications to allocate memory dynamically. This allows the station controller to allocate temporary memory segments for storing data that are needed temporarily. The operating system also provides inter-task communication and data transfer through system mailboxes.

- Local common memory mailboxes (fixed locations) are also used for inter-level data transfer.

3.2.1 Multitasking/Task Synchronization

A brief description of the features of the iRMX86 Operating System used to develop the workstation controller will be helpful in describing how it works. One feature that is used extensively is multitasking. An operating system task (not to be confused with a manufacturing task) is a "program" which performs a specific function. What makes these tasks different from ordinary programs is that they can be invoked by external events in an asynchronous fashion. In fact, these tasks are similar to interrupt handlers except at a higher level. Task level controller and device level controller are examples of operating system tasks. The station controller is basically a collection of these tasks synchronized in a manner that enables it to perform properly. The following are some task characteristics which are defined by the iRMX Operating System:

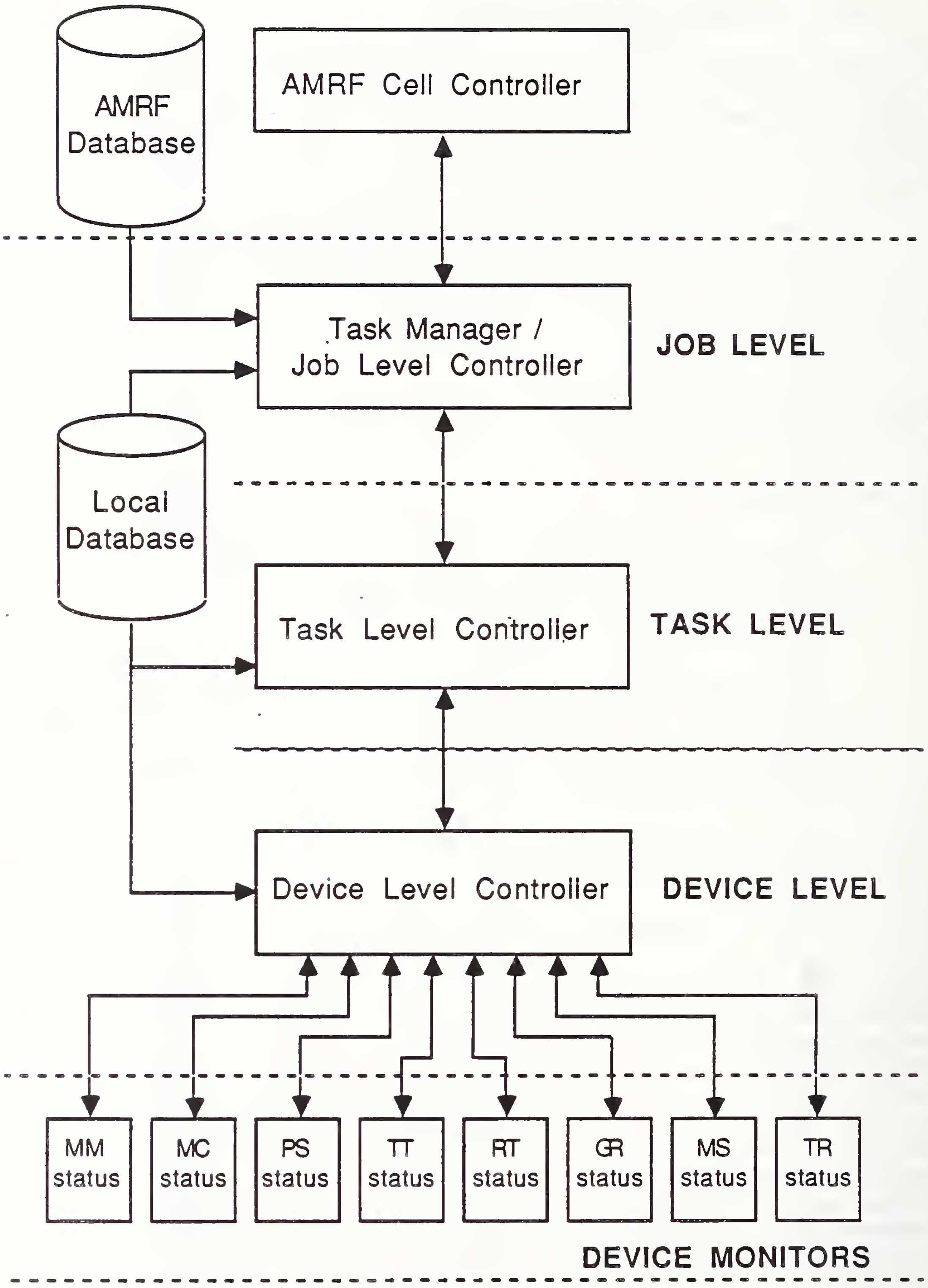


FIGURE III.2. TURNING WORKSTATION CONTROLLER ARCHITECTURE

a) A task has five states (as illustrated in Figure III.3):

- i) Running: the task is executing.
- ii) Ready: the task is ready to execute.
- iii) Asleep: the task is asleep and enters the ready state upon waking.
- iv) Suspended: the task must be resumed by another task before it can enter the asleep or running state.
- v) Asleep-suspended: the task is asleep as well as suspended.

b) Each task has an assigned priority: a higher priority task may preempt a lower priority task, causing the task to exit the running state while it enters the running state.

c) Tasks can communicate with one another by using operating system mailboxes and semaphores. In addition to communication, mailboxes and semaphores are also used for task synchronization. A task is asleep while waiting for information at a mailbox or a semaphore. When the task receives the data that it's waiting for it starts to execute. Therefore the task execution can be synchronized by sending data at proper times.

d) A task may wake up and become ready to run in a variety of ways:

- i) Upon receipt of data from mailboxes or semaphores.
- ii) Upon expiration of time from a self-induced sleep through system call.

e) A task enters the running state when it is the highest priority task in the ready state.

3.2.2 TWS Controller Implementation

The portion of the station's controller which controls the manufacturing process is composed of several tasks arranged in a hierarchy. They are the task manager/job level controller, the task level controller, the device level controller, and the device status modules. The task manager/job level controller is at the top of the hierarchy, the task level controller resides in the second level, the device level controller resides in the third level, and the device monitoring tasks reside in the fourth level.

3.2.2.1 Task manager/job level controller

The task manager/job level controller resides in the highest level of workstation controller hierarchy. It performs several functions.

a) Task management - the task manager activates and synchronizes the execution of the tasks.

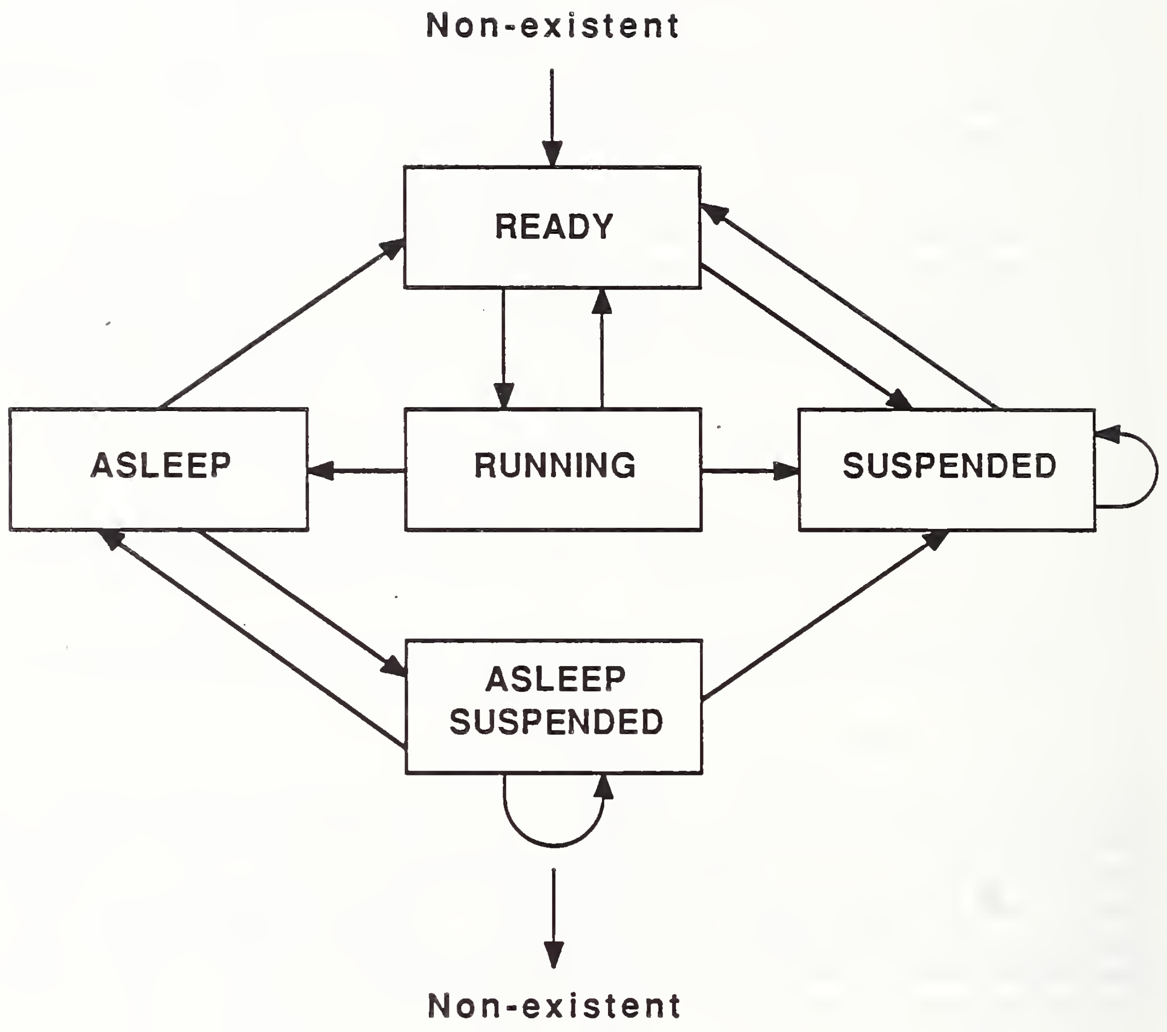


FIGURE III.3. TASK STATE TRANSITION DIAGRAM

b) Manufacturing task decomposition - the task manager decomposes the manufacturing process plan into task level and device level work element sheets and stores them in the local database.

c) User interface - the task manager allows the operator to interact with the workstation controller by activating various user interface routines.

The above functions are performed when the workstation controller is running in either integrated or stand-alone mode (In integrated mode, the workstation functions as a lower level controller to the cell controller. In stand-alone mode, the workstation responds to a local operator command). The task manager also performs the following functions when the workstation is operating in remote mode:

a) Remote command/status interface - the task manager initiates tasks that read command mailgrams from the cell controller via mailboxes and interpret the commands. The task manager also initiates tasks that update the cell controller on the status of manufacturing process execution at appropriate times.

b) Remote data retrieval/update - the task manager initiates tasks that retrieve process plans and other data from the AMRF database. It also initiates tasks that update the AMRF database on the location of resources within the workstation.

3.2.2.2 Task level controller

The task level controller resides in the second level of workstation controller hierarchy. Its main function is to execute the task level work elements (GET_BLANK, LOAD_BLANK, etc.) The following is a summary of functions performed:

a) Task level work element execution - the task level controller scans the task level work element sheet to execute the task level work element in the proper sequence.

b) Activation of the device element level controller - since each work element is broken down into a series of device commands (a device level work element sheet), the device level controller must be activated at appropriate times by the task level controller.

c) Provide update for the task manager - the task level controller updates the task manager on the state of execution of task level work elements. It also updates the cell controller with similar information.

3.2.2.3 Device level controller

The device level controller resides in the third level of workstation controller hierarchy. Its main function is to control the relative sequence in which to send device commands (device elements) out to the device controllers. The following is a summary of functions performed:

a) Control of execution of device elements - by using the device level work element sheets, the device level controller determines the relative sequence in which device commands should be executed and sends out the commands to the device controllers through serial links.

b) Device status processing - the device level controller interprets status feedback from device controllers and determines whether or not the next device command should be sent. The device level controller obtains the status feedback from the devices through device monitoring tasks.

c) Provision of status update for the task level controller - the device level controller updates the task level controller on the state of execution of task level work elements.

3.2.2.4 Device monitoring tasks

The device monitoring tasks reside at the lowest level of the workstation controller hierarchy. Their main function is to communicate with device controllers through the serial links. Each device monitoring task monitors one device. The following is a summary of functions performed:

a) Communication with device controllers in real-time - each device monitoring task waits at a serial port for data from a device controller. When data is received, a device monitoring task interrupts other executing tasks and handles the data received.

b) Status provision for device level controller - each device monitoring task updates the device level controller on the state of execution of the device it is monitoring. Updates are achieved by interrupting the execution of the device level controller and inserting the device status into the status field of the proper device element on the device element sheet.

3.2.2.5 Other tasks

Aside from the main hierarchy of the workstation controller, there are several other tasks that perform functions needed for the workstation to operate. The following is a brief description:

a) Cell communication task - this task handles the protocol involved in communicating with the AMRF cell controller. It also receives command mailgrams from the cell controller and interprets the command. It is used to send status to the cell as well.

b) Database communication task - this task handles the protocol involved in communicating with the AMRF database. It is used to exchange data with the AMRF database.

c) Device test task - this task allows a local operator to test the devices within the workstation individually.

d) Local operation display - this task displays the command and status transfer between the workstation controller and the devices within the workstation. It also displays the job and work elements that the workstation is executing.

e) Local mailbox display task - this task displays the current data being transferred between the devices and the workstation controller.

f) Remote mailbox display task - this task displays the mailgrams exchanged between the workstation controller and the cell controller, as well as the information exchanged between the workstation controller and the AMRF database.

A list of all the tasks is shown in Figure III.4.

3.3 Common Memory

Communication with the cell controller and the AMRF database is achieved via a common memory board that allows remote controllers to grab the system bus and read and write from/to predetermined memory locations. These fixed, predefined locations are termed as "mailboxes". Information exchange is synchronized through read and write locks, and the actual exchange is achieved by reading and writing from/to the mailboxes.

3.4 Database

Two databases are available to the workstation controller: the AMRF database and the local database. The AMRF database is used when the workstation is operating in remote mode, and the local database is used when the workstation is operating in the stand-alone mode. Information exchange with the AMRF database is achieved through the common memory mailboxes. Locally, information exchange with the local database is achieved through operating system calls, since the local database is simply a 40 megabyte Winchester hard disk. Information such as process plan flat files, task level work element sheets, and device level work element sheets are stored in files in the hard disk.

TASK	MODULE	FUNCTION
CM_TASK	CO_MGR.P86	Task manager / job level controller
L1_TASK	TASK1.P86	Task level work element controller
L2_TASK	TASK2.P86	Device level work element controller
CELL_TASK	CELL_MOD.P86	Communication with the AMRF Cell
DB_TASK	DB_MOD.P86	Communication with the AMRF Database
AMPLE_TASK	AMPLE_MOD.P86	Communication with the AMPLE system
DVT_TASK	DVT_MOD.P86	Device test manager
DVT_STA	DVT_STA.P86	Allows device testing to be interrupted
STA_CHG_TASK	STACHG.P86	View or modify various device mailboxes
DSP_TASK	SET_DISP.P86	Sets up various screen displays
LOC_SC_TASK	LOC_MON.P86	Local operation monitor manager
DEVICE MONITORING TASKS:		
MM_TASK	MM.P86	Processes status from the Micromanipulator
MC_TASK	MC.P86	Processes status from the Machine Tool Controller
PS_TASK	PS.P86	Processes status from the Programmable Stop
TT_TASK	TT.P86	Processes status from the Turntable
RT_TASK	RT.P86	Processes status from the Robot
GR_TASK	GR.P86	Processes status from the Gripper Controller
MS_TASK	MS.P86	Processes status from the Malfunction Sensor
TR_TASK	TR.P86	Processes status from the Tray Controller

FIGURE III.4. LISTING OF TASKS THAT COMPRISE THE TURNING WORKSTATION CONTROLLER

4. PROCESS PLAN DECOMPOSITION

In order to produce a part, the device operations within the workstation must be coordinated by the workstation controller. A rather inflexible way of achieving this synchronization is to hard code the workstation controller to execute a specific sequence of operations. This method would require extensive software modification if the workstation is to machine different parts requiring different sequences of device operation. The method used in the workstation controller of the Turning Center is to download a set of manufacturing process data from either the AMRF database or the local database for each part that is to be machined. Using this method, producing different types of parts simply requires a data change. This manufacturing information is stored in "process plan flat files". Prior to the actual production of a part, the workstation retrieves the appropriate process plan flat file and decomposes it into task level and device level work element sheets. After this is done, production can begin. Task decomposition is shown in Figure III.5.

4.1 Process Plan Flat File

Process plan flat files are data used to coordinate the activities within the workstation in order to produce parts. Each part to be made has a process plan flat file associated with it. When the workstation is operating in stand-alone mode, the flat files are retrieved from the local database. When the workstation is in the integrated mode, the flat files are retrieved from the AMRF database. An example of a process plan flat file is shown in Figure III.6.

The process plan flat file is composed of four sections: the header section, the parameters section, the requirements section, and the procedure section. The parameter section is not used by the Turning Workstation.

4.1.1 The Header Section

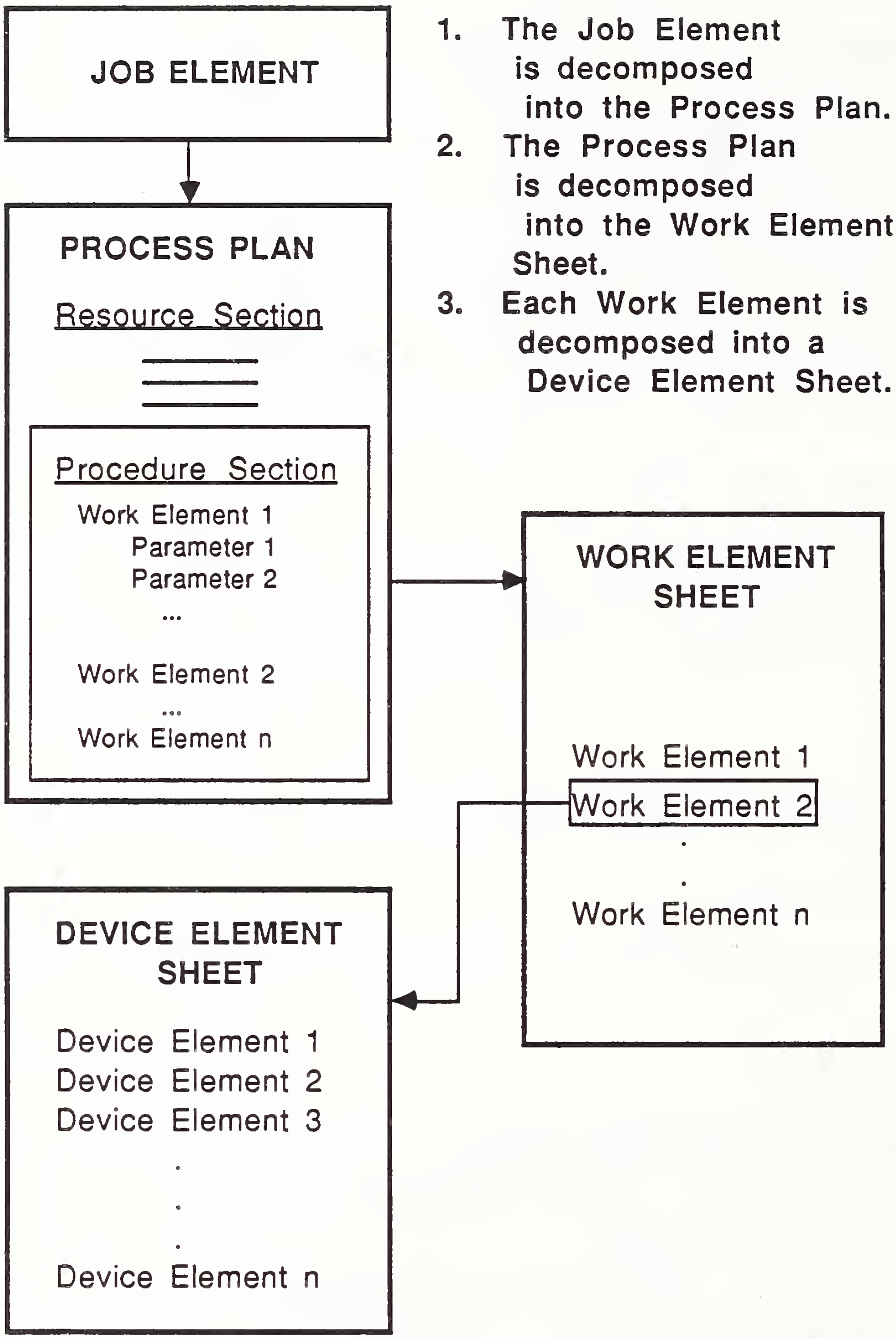
The header section contains elements that identify the process plan. These elements are the Plan Identifier, the Plan Version, the Plan Type, and the Plan Name.

4.1.2 The Resource Section

The resource section of the flat file contains information pertaining to the resources needed by the workstation in order to produce the part in question. These resources include tools, fixtures, fingers, and part blanks.

4.1.3 The Procedure Section

The procedure section of the flat file contains the task level steps that must be taken by the workstation in order to produce the part. This section is composed of a sequence of task level work elements and various parameters. Each task level work element has a set of parameters from which the actual commands of the device level work elements associated with the task level work



1. The Job Element is decomposed into the Process Plan.
2. The Process Plan is decomposed into the Work Element Sheet.
3. Each Work Element is decomposed into a Device Element Sheet.

FIGURE III.5. TASK DECOMPOSITION DIAGRAM

```
-- PROCESS_PLAN --

-- HEADER_SECTION --

PLAN_ID                  := PP_TWS_5;
PLAN_VERSION             := 1;
PLAN_TYPE                := OPERATION_SHEET;
PLAN_NAME                := "BRGSLV";

-- END_HEADER_SECTION --


-- PARAMETER_SECTION --

$$FINGER_ID001           : COL_FINGER;
$$FINGER_ID002           : COL_FINGER;
$$FINGER_ID003           : PART_FINGER;
$$FINGER_ID004           : PART_FINGER;
$$COLLET_ID001           : COLLET;
$$COLLET_ID002           : COLLET;

-- END_PARAMETER_SECTION --


-- REQUIREMENTS_SECTION --

<<1>>   COL_FINGER
            (FINGER_ID          => $$FINGER_ID001);

<<2>>   COL_FINGER
            (FINGER_ID          => $$FINGER_ID002);

<<3>>   PART_FINGER
            (FINGER_ID          => $$FINGER_ID003);
```

FIGURE III.6. PROCESS PLAN FLAT FILE EXAMPLE

```
<<4>>    PART_FINGER
                (FINGER_ID              =>  $$FINGER_ID004);

<<5>>    COLLET
                (FINGER_ID              =>  $$COLLET_ID001);

<<6>>    COLLET
                (FINGER_ID              =>  $$COLLET_ID001);

-- END_REQUIREMENTS_SECTION --




-- PROCEDURE_SECTION --

<<1>>    SFE1_SFE2
                (PREC_STEPS                         =>());

<<2>>    SFE2_SFE3
                (PREC_STEPS                         =>(1));


<<3>>    GET_BLANK
                (RT_PROGRAM                         => /RT101,2/,
                 RT_PROGRAM                         =>/RT103,4/,
                 TRAY_COMPARTMENT                   => 1,
                 PREC_STEPS                         =>(2));

<<4>>    SFE3_SFE2
                (PREC_STEPS                         => (3));

<<5>>    SFE2_SFE1
                (PREC_STEPS                         => (4));
```

FIGURE III.6. PROCESS PLAN FLAT FILE EXAMPLE(CONT'D.)

```
<<6>>   LOAD_BLANK
            (RT_PROGRAM                      => /RT110,3/,
             PS_PROGRAM                      => /glxxx,1/,
             PREC_STEPS                      => (5));

<<7>>   MACHINE
            (PART_PROGRAM                    => /BRGSLV1,6/,
             PUSH_PROGRAM                    => /PBRGSLV1,2/,
             MS_CMD                          => s,
             PS_PROGRAM                      => /gl175,7/,
             PREC_STEPS                      => (6));

<<8>>   SFE1_SFE2
            (PREC_STEPS                      => (6));

<<9>>   GET_BLANK
            (RT_PROGRAM                      => /RT101,2/,
             RT_PROGRAM                      =>/RT104,4/,
             TRAY_COMPARTMENT                => 2,
             PREC_STEPS                      =>(8));

<<10>>  SFE3_SFE2
            (PREC_STEPS                      => (9));

<<11>>  SFE2_SFE1
            (PREC_STEPS                      => (10));

<<12>>  REM_FIN_PT
            (RT_PROGRAM                      => /RT400,2/,
             PREC_STEPS                      => (11,7));
```

FIGURE III.6. PROCESS PLAN FLAT FILE EXAMPLE(CONT'D.)

```
<<13>>  MACHINE
                (PART_PROGRAM                  => /BRGSLV1,6/,
                 PUSH_PROGRAM                  => /PBRGSLV1,2/,
                 MS_CMD                        => s,
                 PS_PROGRAM                    => /gl175,7/,
                 PREC_STEPS                    => (12));

<<14>>  SFE1_SFE2
                (PREC_STEPS                    =>(12));

<<15>>  SFE2_SFE3
                (PREC_STEPS                    =>(13));

<<16>>  DRP_OFF_PT
                (RT_PROGRAM                    => /RT201,2/,
                 RT_PROGRAM                    => /RT103,4/,
                 PREC_STEPS                    => (15));


<<17>>  SFE3_SFE2
                (PREC_STEPS                    => (16));

<<18>>  SFE2_SFE1
                (PREC_STEPS                    => (17));

<<19>>  REM_FIN_PT
                (RT_PROGRAM                    => /RT400,2/,
                 PREC_STEPS                    => (18,13));

<<20>>  SFE1_SFE2
                (PREC_STEPS                    =>(12));
```

FIGURE III.6. PROCESS PLAN FLAT FILE EXAMPLE(CONT'D.)

```
<<21>>  SFE2_SFE3
                (PREC_STEPS                     =>(13));

<<22>>  DRP_OFF_PT
                (RT_PROGRAM                     => /RT202,2/,
                 RT_PROGRAM                     => /RT104,4/,
                 PREC_STEPS                     => (15));


-- END_PROCEDURE_SECTION --

-- END_PROCESS_PLAN --
```

FIGURE III.6. PROCESS PLAN FLAT FILE EXAMPLE(CONT'D.)

element can be derived, thus allowing the task level work elements to be generic. In actual operation, the content of the procedure section is further translated into a format that is computer legible: the procedure section is decomposed into the task level and device level work element sheets.

4.2 Task Level Work Element Sheet

The task level work element sheet is decomposed from the procedure section of the process plan flat file. It contains a series of task level elements. Each task element contains information needed by the workstation controller in order to execute the manufacturing process.

4.2.1 Format for The Task Elements

Each task element has the following fields:

a) Step number: indicates the relative sequence of execution of the task element

b) Precedence numbers: step number of task elements that must by completed prior to the task element's execution

c) Task element name: GET_BLANK, LOAD_BLANK are examples. These names are used to load the proper device element sheets.

d) Status field: execution status of the work element

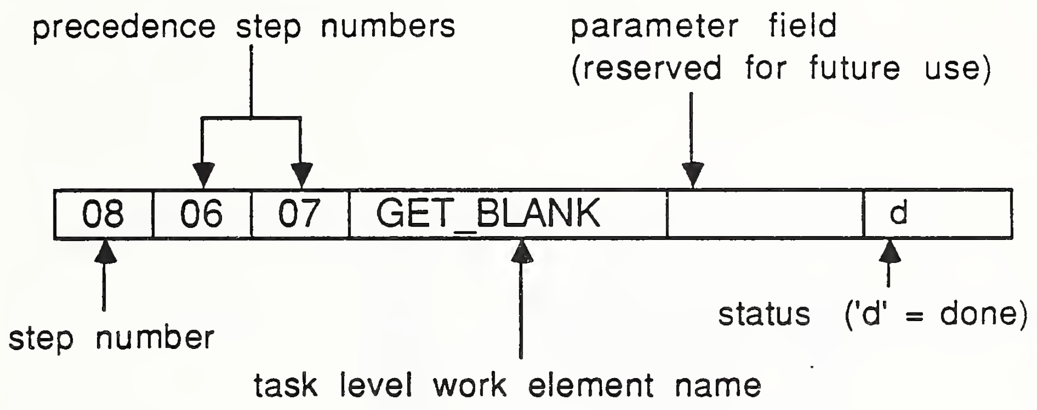
Format for the task elements and an example task element sheet are shown in Figure III.7 and Figure III.8, respectively.

4.2.2 Execution of The Task Elements

The task elements are executed by the task level controller of the workstation controller. The task level controller looks at each element of the task element sheet to see if it is ready to begin execution. A task element can begin execution only if all of its precedent steps have completed their execution. The state of a task element's execution is indicated at the status field of the task element. The status can take on three values: NULL, which indicates that the task element has not been executed yet, BUSY, which indicates that the task element is currently executing, and DONE, which indicates that the task element has completed its operation. The status is provided by the device level controller. With these parameters, the task level controller can determine which task elements can begin execution and which ones cannot. This process is shown on Figure III.9.

4.3 Device Level Work Element Sheets

The device level element sheets contain a series of device elements. Each device element contains information needed for the device level controller of the workstation controller to execute device operations.



Format:

step number (2 ASCII digits: 2 bytes)
precedence step numbers (8 ASCII digits: 4 numbers, 8 bytes)
work element name (ASCII string up to 13 characters)
parameter field (13 bytes: reserved for future use)
status field(4 bytes: ASCII characters)

FIGURE III.7. TASK LEVEL WORK ELEMENT FORMAT

0100	GET_BLANK
0201	LOAD_BLANK
0302	MACHINE
0402	GET_BLANK
050304	REMOVE_LOAD
0605	MACHINE
0705	UNLOAD_PART
080607	REMOVE
0908	UNLOAD_PART

FIGURE III.8. AN EXAMPLE OF A TASK LEVEL WORK ELEMENT SHEET

0100	GET_BLANK	done
0201	LOAD_BLANK	done
0302	MACHINE	busy
0402	GET_BLANK	done
050304	REMOVE_LOAD	null
0605	MACHINE	null
0705	UNLOAD_PART	null
080607	REMOVE	null
0908	UNLOAD_PART	null

The task level controller scans the above task level work element sheet to see which task elements can be executed. Suppose the task level controller is examining the 5th task element (REMOVE_LOAD) to see whether or not it should be executed. The task element's precedent steps, step3 (MACHINE) and step 4 (GET_BLANK) are examined. GET_BLANK has been completed, but MACHINE has not. Therefore REMOVE_LOAD cannot begin execution yet.

FIGURE III.9. EXECUTION OF TASK LEVEL WORK ELEMENTS

4.3.1 Format/Fields for Each Equipment Element

Each device element has the following fields:

a) Step number: indicates the relative sequence of execution of the device element

b) Precedence numbers: step numbers of device elements that must be completed prior to the device element's execution

c) Device name: indicates which device is to receive the command. Each name has two characters. "RT" for robot and "GR" for gripper are examples.

d) Command: actual command sent to the device. The command is usually a string of characters.

e) Status: execution status of the device command

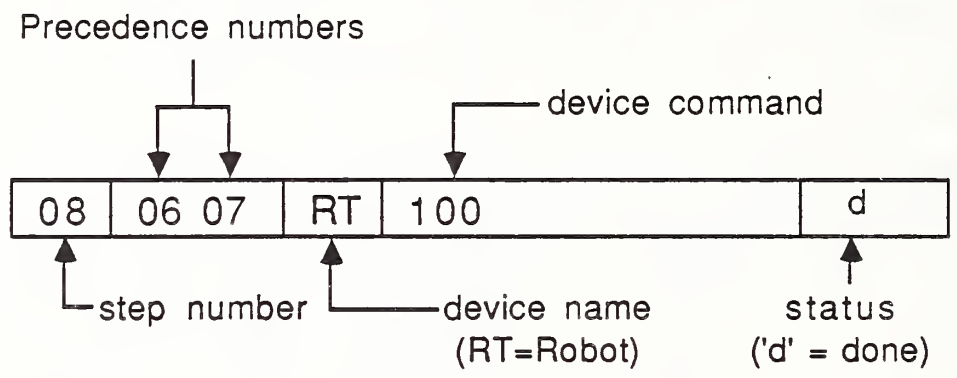
The format for the device level work sheet and an example device element sheet are illustrated in Figure III.10 and Figure III.11, respectively.

4.3.2 Execution of the Device Elements

The device elements are executed by the device level controller of the workstation controller. The manner of execution is similar to that of the task elements. The device level controller looks at each element of the device element sheet to determine if it is ready to commence execution. A device element can begin execution only if all of its preceding steps have completed their execution. The state of a device element's execution is indicated at the status field of the device element. The status can take on three different values: NULL, which indicates that the device element has not begun execution yet, BUSY, which indicates that the device element is currently executing, and DONE, which indicates that the device element has completed its operation. The status for each device is provided by the device monitoring tasks. With these parameters, the device level controller can determine which device elements can begin execution and which ones cannot. Device element execution is illustrated in Figure III.12.

The workstation is capable of executing up to five task level work elements at the same time. This means that the device level controller must be able to execute device elements from up to five device element sheets at the same time. What actually occurs is that the device level controller scans through each of the device element sheets and stores the commands to be sent to the devices in a command buffer. From there the commands are sent to the devices after all the device element sheets have been scanned.

When a task level work element is to execute, the corresponding device level work element sheet is loaded from the local database into random access memory.



Format:

step number (2 ASCII digits: 2 bytes)
precedence numbers (8 ASCII digits: 4 numbers, 8 bytes)
device name (2 ASCII characters)
device command (10bytes: ASCII string)
status field(4 bytes: ASCII characters)

Device names:

MM	= Micromanipulator	MC	= Machine Tool Controller
PS	= Programmable Stop	TT	= Turntable Controller
RT	= Robot Controller	GR	= Gripper Controller
MS	= Malfunction Sensor	TR	= Tray Controller

FIGURE III.10. DEVICE LEVEL WORK ELEMENT FORMAT

0100	RT100
0201	MMj
0302	GRc2
0402	MMh
050304	RT102

FIGURE III.11. AN EXAMPLE OF A DEVICE LEVEL WORK ELEMENT SHEET

0100	RT001	done
0201	MMj	busy
0302	GRc1	null
0403	RT002	null

The device level controller scans the above device level work element sheet to determine which device elements can be executed. The algorithm to do this is completely analogous to that of the task level controller. Suppose the device level controller is examining the 3rd device element (close gripper 1) to see if the command can be sent to the gripper. Its precedent step (step 2 - slide the micromanipulator) has not finished yet (note the busy status). Therefore step 3 cannot be executed at this time.

FIGURE III.12. EXECUTION OF DEVICE LEVEL WORK ELEMENTS

The execution of the task element then is simply the execution of the device level work element sheet. The task level work element name is actually used as the filename of the corresponding device level work element sheet.

4.4 Decomposition of Process Plan Flat Files Into Task Level And Device Level Work Element Sheets

Each step, or task level work element, of the procedure section contains a set of parameters. These parameters are used to determine the precedence of the task element, as well as the exact device commands that are associated with the task element. The use of parameters allows the task level work elements to be more generic. The following example illustrates this: suppose the workstation is to execute a GET_BLANK task level work element. The robot is to pick up a part blank at the buffer tray, but the blanks are stored in different compartments of the tray. This means that on different occasions, different robot programs need to be executed in order to pick up blanks at different tray compartments. Instead of using different GET_BLANK tasks, parameters that specify which tray compartment the robot is to go to are used. This way the same task element can be executed, even though the actual robot commands are different.

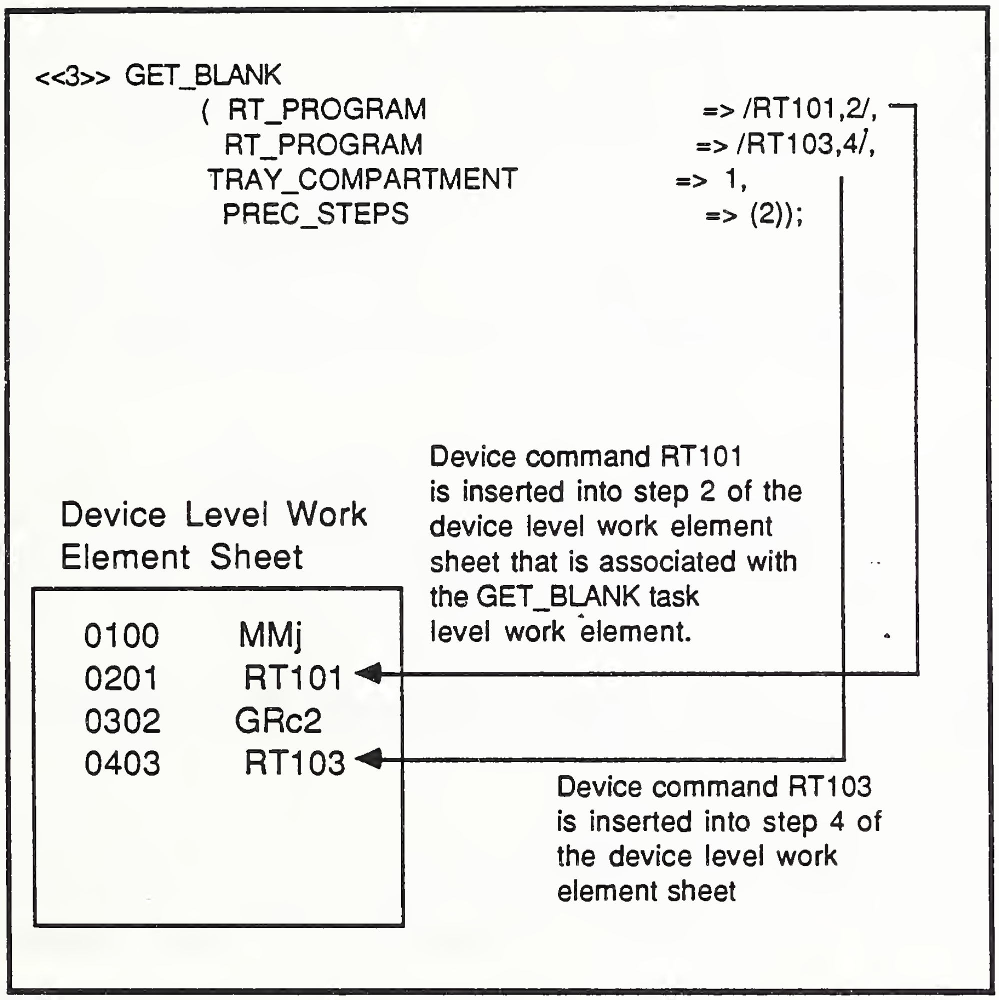
A general device level work element sheet is associated with each task level work element. Steps within the device level work sheet are changed in accordance with the parameters associated with the task level work element. A parameter is composed of a parameter name and the actual parameter. Parameters that affect the device elements are composed of the actual device command and the step number of the device element. The step number allows the workstation controller to easily insert the proper device command into the proper location of the device element sheet. This is illustrated in Figure III.13.

5. REMOTE COMMUNICATION

The Turning Workstation communicates with the rest of the AMRF in a variety of ways. Communication with the AMRF cell controller and database is achieved through the AMRF Network and the common memory board installed within the TWSC. Communication with the Automated Manufacturing Programming Language Environment (AMPLE) system is achieved through an RS232 serial interface and the AMRF Network (see section 5.3).

5.1 Cell to Turning Workstation Communication

When TWS is operating in integrated mode, it receives manufacturing commands from the AMRF cell controller instead of a local operator. Communication is achieved through the common memory and the AMRF network. Command mailgrams are sent by the cell and status mailgrams are returned to the cell by TWSC.



<<3>> GET_BLANK
(RT_PROGRAM => /RT101,2/,
RT_PROGRAM => /RT103,4/,
TRAY_COMPARTMENT => 1,
PREC_STEPS => (2));
Device Level Work Element Sheet
0100 MMj
0201 RT101
0302 GRc2
0403 RT103
Device command RT101 is inserted into step 2 of the device level work element sheet that is associated with the GET_BLANK task level work element.
Device command RT103 is inserted into step 4 of the device level work element sheet

Figure III.13. PROCEDURE SECTION PARAMETERS

5.1.1 Cell to TWS Communication Protocol

Communication between the AMRF cell controller and the Turning Workstation is achieved through the use of mailboxes in common network memory. A mailbox is dedicated to the command mailgrams sent by the cell, and another mailbox is used to transfer status mailgrams to the cell. Software read and write locks and sequence variables are used to synchronize and identify the data that is transferred. The following is a more specific description:

READ LOCK: A 2-byte variable used to determine whether or not data can be read from the mailbox. If it is set, then neither the cell nor the Turning Workstation can read the data. If it is not set, then data reads can commence. Typically, it is set by either controller when it is about to write data into the mailbox and reset when data writing is done.

WRITE LOCK: A 2-byte variable used to determine whether or not data can be written into the mailbox. If it is set, then neither the cell nor the Turning Workstation can write data into the mailbox. If it is not set, then data writing can commence. Typically, it is set by either controller when it is about to read data from the mailbox and reset when data read is done.

SEQUENCE NUMBER: A 2-byte variable used to indicate the sequence of the data transfer in question. It is initialized when the workstation becomes ready to accept commands from the cell and updated each time a data transfer occurs.

LENGTH: A 2-byte variable used to indicate the byte length of a data mailgram.

5.1.2 Cell to Turning Workstation Command/Status Mailgram Formats

The communication between the AMRF cell controller and the Turning Workstation is achieved by passing command and status mailgrams through the network. The following is a description of the protocol for the mailgrams.

5.1.2.1 Command Mailgram

When the cell is to activate the Turning Workstation, it sends a command mailgram to the workstation through a common memory mailbox. The mailgram is composed of the header section, the transition command section, and the order action record section. The header section contains identifying elements for the mailgram such as the header and the command number. The transition command section is used by the cell to control the state of the workstation. Typical commands are WARM_STARTUP and COLD_SHUTDOWN. The order action record contains the manufacturing action that is to be taken by the workstation. Typical order actions are RECEIVE_TRAY and MACHINE_LOT. In addition to the manufacturing action, the order action record also contains parameters such as the process plan id of the process plan that is to be executed by the workstation. Figure III.14 shows the composition of a typical command mailgram in more detail.

5.1.2.2 Status Mailgram

The status mailgram (see Figure III.15) is used by the Turning Workstation to update the AMRF cell controller on the status of execution of the corresponding command. The mailgram is composed of the header section, the length indicator section, the echo section, the transition status section, and the order action status section. The header section contains identifying elements for the mailgram such as the header and status number. The length indicator section indicates the length of various fields within the status mailgram. The echo section simply echoes the text of the command that is being executed. The transition status section indicates the status of execution of a transition command. The order action status section contains the status of execution of the order that is being filled by the workstation. It may contain parameters such as error conditions or number of parts machined.

5.2 Database Communication Module

Communication with the AMRF database is done by the Turning Workstation (TWS) in order to retrieve process plans and reports needed for a cell-specified operation and to update the status of the workstation's operation for the cell. The data received from the AMRF database includes the process plan (in flat file form), the Lot Status report, the Tray Definition report, and the Tray Contents report. The reports that are updated to the AMRF database are the Workstation Item Action report and the Equipment Item Action report.

Information that is obtained from the selected reports includes, for each part, the item number, the item name, the sector number, the origin of sectors in a tray, the input tray, the in-tray workstation, the output tray, the out-tray workstation, and the kit order identification. Of these items, the item name and item number are used in the update reports.

The Equipment Item Action and the Workstation Item Action reports are used to update the name and location of a part as each stage of its machining is completed. For example, a part that is made by TWS is the nipple connector. It is received in a tray with the name "NIPPLERAD_BT". After its first stage of machining is completed, it is called "NIPPLERAD_IT". When the part is finished, it is then called "NIPPLERAD_FT" and the Equipment Item Action report is updated a third and final time for its name and location. After the finished part is placed in the tray, ready to be shipped, the Workstation Item Action report is updated for the location of the finished part. This report specifies the item number and its location (e.g. "T_TABLES" is used).

In addition to selecting process plans from the AMRF database, process plans can be inserted into the database. Although a separate communication module is used for this operation (for communication testing), it follows the same command and status formats of the main AMRF database communication module.

Field	Length (bytes)	Entry or Possible Value
command message identifier	4	'CMDF'
command length	2	72
command number	2	nn
command time stamp	22	'YYYY,DDD,HH,MM:ss:DDDb'
transition action length	2	36
order action length	2	00
resource allocation length	2	00
transition action update	2	nn
transition action keyword	16	'SYNC' or 'WARM_STARTUP'
transition parameter count	2	01
transition parameters	16	transition parameters
number of order actions	02	01
order action record		
order id	16	
update number	16	
action	16	'EXECUTE'
job level work element	16	SHIP_TRAY
number of parameters	16	3

The length of the remaining parameter fields is based upon the number of parameters specified

parameter 1 name	16	'WS_ID'
parameter 1 value	16	'TWS'
parameter 2 name	16	'TRAY_ID'
parameter 2 value	16	tray id string
parameter 3 name	16	'TRAY_TYPE'
parameter 3 value	16	e.g. '3_SECTOR_TRAY

FIGURE III.14. CELL TO TWS COMMAND MAILGRAM PROTOCOL

Field	Length (bytes)	Entry or Possible Value
status message identifier	4	'FDBF'
status length	2	276
status number	2	nn
status time stamp	22	'YYYY,DDD,HH:MM:SS:mmmb'
echo command length	2	24
transition status length	2	36
order status length	2	176
resource requests length	2	00
echo command number	2	nn
echo command time	22	'YYYY,DDD,HH:MM:SS:mmmb'
echo command segment		
transition status version	2	nn
transition status keyword	16	'SHUTDOWN', 'READY'
transition status parameter count	2	01
transition status parameter	16	blank
number of order status records	2	01
order status record		
order id	16	order id string
update number	16	ASCII digits
status	16	status strings
echo work element	16	job level work element string
number of parameter pairs	16	ASCII digits

The number of remaining parameter fields depends on the number of parameter pairs specified.

parameter 1 name	16	e.g. 'ERROR_CONDITION'
parameter 1 value	16	e.g. ASCII string
parameter 2 name	16	e.g. 'COMPLETED'
parameter 2 value	16	e.g. ASCII digits
parameter 3 name	16	e.g. 'SCRAPPED'
parameter 3 value	16	e.g. ASCII digits

FIGURE III.15. TWS TO CELL STATUS MAILGRAM PROTOCOL

The aforementioned reports are obtained from the database using DML (Data Manipulation Language) transactions. Protocol for DML transactions and AMRF database/TWS communications is discussed in the following section.

5.2.1 AMRF Database/TWS Communication Protocol

As with any handshaking protocol, AMRF database/TWS communications require a command and a status. Here, the Turning Workstation gives commands and the database returns statuses.

Before any requests for data are made of the database, communication is begun by using the reconfiguration commands "ABORT", "INITIATE", and "START-UP", in this order. After these commands are successfully executed DML transactions can be done. When TWS wishes to end communication with the database, the reconfiguration commands "SHUTDOWN" and "TERMINATE" are sent to the database.

In order for the commands to be recognized as new commands by the database, each command has a command number and sequence number that are incremented each time a new command is sent. The initial command/sequence number that TWS starts with is "1" with the "ABORT" command.

A status message from the database accompanies each TWS command. In order for TWS to know that it is the correct status for a certain command, it must check the command number and compare it to the command number of the latest command, as well as making sure that it is a status report from the database. Checking for the Data Server status indicator accomplishes this. The appropriate status for a particular command is then checked. For the reconfiguration commands, the database status is repeatedly checked until the correct status is found. When it is found, the next command is issued to the database. In the case of DML requests, if an error is found, the request will be made again for up to two times if needed.

The types of status that are expected from the database for a successful transaction vary for each different command. For the "ABORT" command a "D" (for "down") is expected. An "S" (for "sync") must be seen for the "INITIATE" or "SHUTDOWN" reconfiguration commands. Before the database can process requests an "R" (for "ready") must be received for the "STARTUP" command. After the "STARTUP" has been successfully executed, the following DML transactions are looked statuses at the summary-status field of the status report. Possible statuses are a "B" (operation still pending), a "D" (operation completely successful), or a "C" (successful initiation of operation). This is unlike the reconfiguration commands whose statuses appear in the server status field. The status report format will be discussed below.

All communication is done using common memory locations that are previously assigned for each workstation. There are three mailboxes for AMRF database/TWS communication. The commands to the database are entered into the mailbox named DS_TWS_CMD which has address ODD200H and a size of 256 bytes. The statuses from the database are received from the mailbox named DS_TWS_STS at location ODD400H. It has a maximum length of 256 bytes. Any data that is requested from or is sent to the database is put into the mailbox DS_TWS_D1 with a maximum length of 8 kilobytes and a starting address of 0DD600H. Whenever a mailbox is specified in a DML transaction, the name of the mailbox is used.

5.2.2 AMRF Database Command And Status Report Formats

All the AMRF database commands follow the same basic format. The first fourteen bytes of any command, reconfiguration, DML, or CANCEL commands are fixed, comprising the "fixed segment". The following bytes, making up the transaction string (the "variable segment"), however, will be in one of two forms, depending on its length. The first form, form "A", is for a transaction string of length less than 128 bytes long. Reconfiguration commands and DML requests with DML strings less than 118 bytes long use this form. Commands with DML strings 118 bytes long or longer use form "B".

The fixed segment of a command includes information such as the user identification, command number, the fixed segment length, and the fixed segment prefix. Also included are the length of the entire command, minus the first two header bytes, and an indicator that confirms that the command string is a database command.

The variable segment, the transaction string, which is formatted in one of the two forms, indicates the type of command that is to be executed. Other information that is included are the application type, transaction identifier, and optional arguments such as the DML strings. Accompanying these items are their lengths and, of course, the length of the transaction string (less the header bytes). The command format is shown graphically in Figure III.16.

Unlike the command formats, the status reports have one form composed of a fixed segment and a variable segment. It is within the fixed segment that the statuses for reconfiguration commands are looked for. Other information included here are the identifications of the user and the database; the report time and number; the last command number; and the status of the server, user-link, and master-link. It is in the server-status field that the reconfiguration command status is found. Ending the fixed segment is a pad-byte composed of a null (i.e. 00H).

The variable segment consists of a twelve byte fixed structure followed by a variable six byte segment and optional fields. Items that comprise the first twelve bytes are the header byte, transaction status indicator; the transaction status length, less the header bytes; the transaction identification and its length; an echo of the user transaction identification and commands type; and a status-element type indicator accompanied by its length. The bytes that follow include the summary-status, a pad-byte, the detail-status, and an optional field. A status resulting from a DML transaction would be found in the summary-status field and an error message may be found in the optional field that describes the error. A graphic representation of the database status report format is shown in Figure III.17.

Byte Number	Value	Comments
1	ACH	DataServer Command Indicator
2	*	Command Length Indicator
3	80H	Fixed Segment Prefix
4	0AH	Fixed Segment Length
5	*	Command Number, MSB first
6	TWSC	User ID in ASCII, Left Justified
Form "A":		
15	30H	Universal type 16, structured
16	*	Length of String in Bytes, 8H for reconfiguration commands
17	*	Transaction Type, possible values are: 81H = Initiate, 82H = Startup, 83H = Shutdown, 84H = Terminate, 85H = Abort, 86H = DML, 87H = Cancel
		Length of Transaction Type
18	00H	Application type 16 (transaction ID)
19	50H	Length of Transaction ID
20	04H	Transaction Identification,
21-24	*	values are: INIT for Initiate command, STUP for Startup command, SHDN for Shutdown command, TERM for Terminate command, ABRT for Abort command, DML for DML command.
25-...	*	Optional arguments, e.g. DML strings, including length and data
Form "B":		
15	30H	
16	82H	Indicates string length in 2 bytes
17-18	*	Length of string, 2 byte integer
19	*	Transaction type, same as above
20	00H	Length of transaction type
21	50H	Application type 16
22	04H	Length of transaction ID
23-26	*	Transaction identifier, as above
27-...	*	Optional arguments, as above

NOTE: Values indicated by an asterisk are dependent on the particular command.

FIGURE III.16. DATABASE COMMAND FORMAT

Byte Number	Value	Comments
1	0A1H	DataServer Status Indicator
2	*	Length of status report
3	80H	Context type 0 (fixed segment)
4	20H	Length of Fixed Segment
5-12	TWSC	User ID
13-20	*	Server ID
21-28	*	Report Time
29-30	*	Report Number
31-32	*	Last Command Number
33	*	Server-status
34	*	User-link-status
35	*	Master-link-status
36	00H	Pad-byte
37	0A1H	Context type 1
38	*	Length
Transaction Status Report:		
39	0A0H	Transaction Status Indicator
40	*	Transaction Status Length
41	81H	Transaction ID
42	*	Transaction ID Length
43-46	*	User Transaction ID Echo
47	*	Command Type Echo
48	00H	Command Length
49	82H	Status-elements Type Indicator
50	06H	Status-elements Length
51	*	Summary-status
52	00H	Pad-byte
53-56	*	Detail-status
Optional fields:		
57	*	Field type, 83H = error message, 84H = data reference, 85H = byte count, 86H = row count, 87H = source station.
58	*	Length
59	*	Value

NOTE: Values indicated by an asterisk are dependent on the particular command. The length bytes used here follow the convention for the short form. Other length encodings are described in the text.

FIGURE III.17. DATABASE STATUS REPORT FORMAT

As a last note on the command and status formats, the length of each and their fixed and variable segments are entered in a different manner depending on their lengths. If the length of a segment is less than 127 bytes long (7FH), then the length field will only require one byte and is directly recorded in that byte. If the length is greater than 127 bytes then more than one byte will be needed for the length field. In this case the first byte in the field will indicate that the bytes that follow will be the length bytes. This byte is represented by "8xH" where "x" is a "1", or a "2", etc., which indicates the number of bytes that follow are used for the length. So then an "82H" as a first byte in the length field means that the following two bytes contain the length of the corresponding segment, command or status and that the total number of bytes that comprise the length field is three.

As an alternative to the above encodings, a "mark byte" may be used to mark the beginning and the end of the data in the segment without regard to the length of the data in the segment. In the place of the length field, an "80H" is entered. The data in the segment then follows and is terminated by two "end-of-data" bytes which are both nulls.

The convention used at TWS for database commands is to explicitly encode the length of the segment in the length fields, using the short and long forms as needed. In the section that follows, the convention using the "mark-bytes" is not illustrated.

5.2.3 DML Strings Used by TWS

AMRF database accesses are accomplished by using DML strings in commands to specify what is needed. The DML strings that are used to request reports from the database are as follows (where items in lower case are to be entered as needed):

for the Process Plan:

```
SELECT PLAN_TEXT FROM PROCESS_PLAN WHERE
EXEC_SYSTEM = 'TWS' AND PLAN_ID = ' process_plan_id '
AND PLAN_VERSION = plan_version USE MEMORY 'DS_TWS_D1'
FORMAT 'G{A}';
```

for the Lot Status report:

```
SELECT * FROM LOT_STATUS WHERE LOT_ID = lot_id
USE MEMORY 'DS_TWS_D1';
```

for the Tray Definition report:

```
SELECT * FROM TRAY_DEFINITION WHERE ITEM_NAME_T =
' item_name ' USE MEMORY 'DS_TWS_D1';
```

for the Tray Contents report:

```
SELECT * FROM TRAY_CONTENTS WHERE CONTAINER_NR =
' container_number ' USE MEMORY 'DS_TWS_D1';
```

for the Equipment Item Action report:

```
UPDATE EQUIP_ITEM_ACTION SET * = USE MEMORY 'DS_TWS_D1'
WHERE ITEM_NR = ' item_number ';
```

and, finally, for the Workstation Item Action report:

```
UPDATE WS_ITEM_ACTION SET * = USE MEMORY 'DS_TWS_D1'
WHERE ITEM_NR = ' item_number '.
```

In order to insert a process plan to the database, TWS uses this DML string:

```
INSERT INTO PROCESS_PLAN (EXEC_SYSTEM,PLAN_ID,
PLAN_VERSION,PLAN_TEXT) VALUES ('TWS',' process_
plan_id ',' process_plan_version ',USE MEMORY
'DS_TWS_D1' FORMAT 'G(A)';
```

The "item_number", "item_name", "process_plan_id", "container_number" (also called the "tray_serial_number") are ASCII character strings with a maximum length of 16 bytes. The "process_plan_version" and the "lot_id" are integer values written in ASCII in the DML string and are of a maximum length of four numerical characters. An asterisk, "*", within a string has the meaning of "all" which follows the DML conventions.

5.3 The AMPLE Communication Module

The Automated Manufacturing Programming Language Environment (AMPLE) system is used to verify the validity of a process plan[1] and to animate workstation operations at the Turning Workstation (TWS) as well as at other workstations [13,14]. By using the TWS/AMPLE communication module, the TWS can send process plans, accompanying equipment instruction lists, and cutter location files to the AMPLE system. The AMPLE system will, in turn, check and return status messages on the state of the data sent by TWS.

5.3.1 Function of AMPLE Within The Turning Workstation

Two functions that AMPLE executes for the Turning Workstation are to check numerical control (NC) programs, and to check robot programs. In order to check NC programs, AMPLE uses the cutter location files sent by TWS. To check robot programs, AMPLE uses the equipment instructions received from TWS and

[1]A process plan is valid if it conforms with the AMRF flat file form, if the work elements listed in the procedure section are performable by TWS, and if all work elements needed to make a part are in the correct sequence.

compares the robot programs in the instructions to those that were provided ahead of time from the programmer of the robot programs. The equipment instructions are a translated version of the process plan in flat file form. The process plan is created either by the Process Planning subsystem, and entered into the AMRF database, or by the Turning Workstation operator, and uploaded into the AMRF database.

Another function that AMPLE performs is to animate overall workstation operations. It does this through animation on a CRT, via Silicon Graphics, of the work elements in the translated process plan provided by TWS. This allows for a visual checking of workstation operations.

Before creating an animation of the TWS operation, information must be provided to AMPLE ahead of time. Some of this information includes dimensional data. The dimensional data describes the volume in which a component exists. The volume describes the general shape of the component and includes any critical parts that may conflict with another component. This data is then used to determine a graphic representation of each component which is used in the animation. Components include the robot, the gantry, the turntable, and the turning machine.

In order to animate the components, software subroutines are written for each component. The software describes the movement of each component in a correct operation. In addition to the software, the Cartesian coordinate position of the tool tip and the gripper position vector for all robot moves are calculated. After all the information that is needed is transferred to AMPLE, communication between the two systems is done as described in the next section.

5.3.2 Communication Protocol

5.3.2.1 Commands And Status

The TWS/AMPLE protocol consists of an ASCII string command from TWS (or AMPLE) and a status of "READY" from AMPLE (or TWS). The "READY" string is used to acknowledge some of the commands sent. Not all of the commands are acknowledged by a "READY". The commands are used to signal the beginning of a session, to indicate the type of data that will be transmitted, and to mark the end of a transmission of data.

5.3.2.2 Data Transfer Between TWS and AMPLE

Sets of data transmitted between the TWS and AMPLE systems include the process plan, its corresponding equipment instructions, the cutter location file, and status messages. The first three sets are sent by TWS and the status messages are sent by AMPLE.

Items that comprise a set of data from TWS are the process plan text (or equipment instructions or cutter location file), the number of bytes transmitted, and the checksum. The process plan is the original version retrieved from the database and the equipment instructions list is the TWS-translated version. The translation is done separately by the Turning Workstation Controller. Both the process plan and the equipment instructions are stored in a file in the local database beforehand. The number of bytes is the byte count of the text sent and the checksum is the sum of the ASCII values of all the bytes within the text and is of word length.

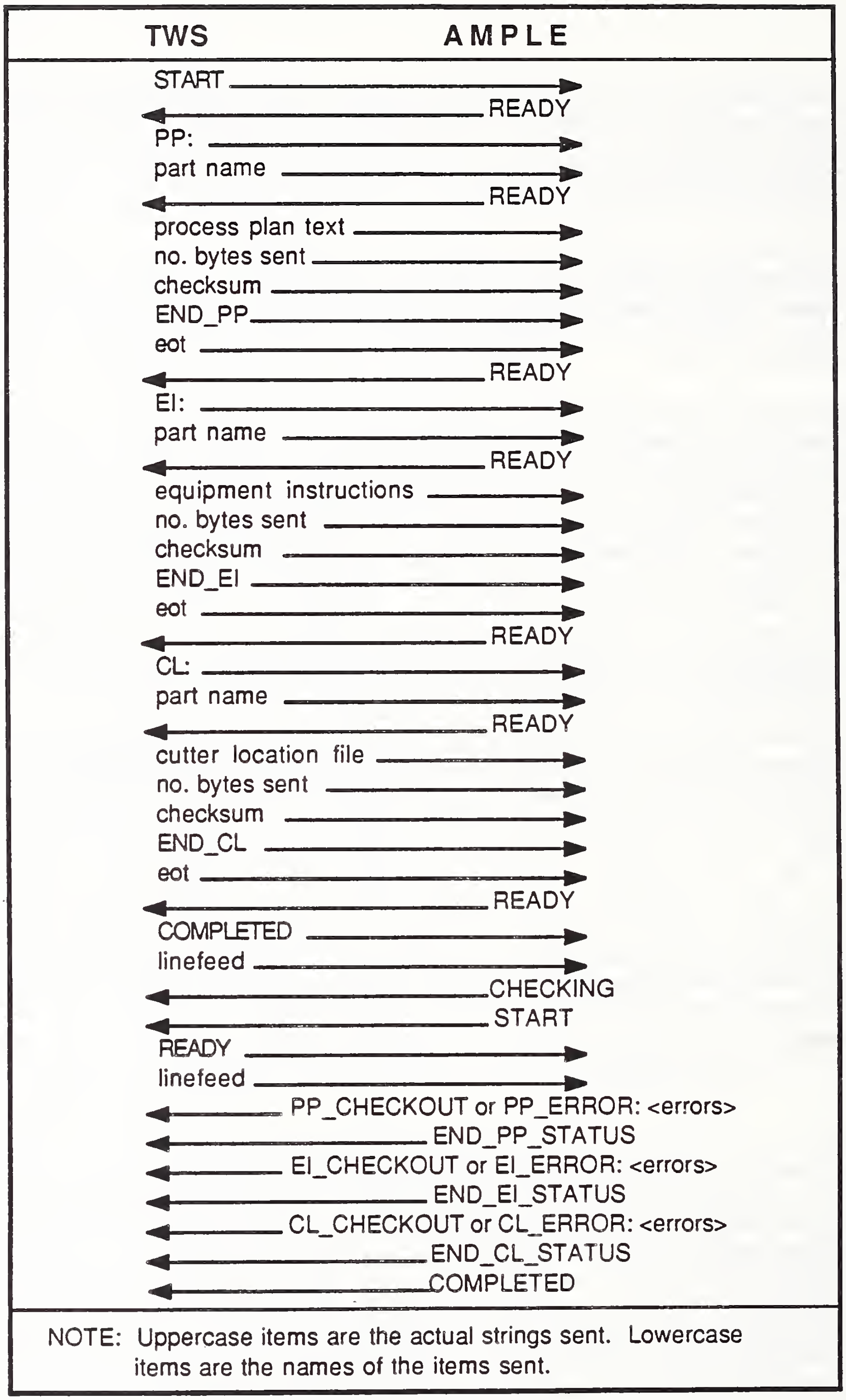
To begin a session with AMPLE, the TWS would send a "START" command to AMPLE. This is then acknowledged with a status of "READY" from AMPLE as discussed above. Next, the three sets of data will be sent, using the same format for each, in the order of process plan text, equipment instructions, and then cutter location file.

The format for the three sets of TWS data is as follows. A three byte string consisting of two uppercase letters and a colon is sent first depending on the text. The string would be "PP:","EI:", or "CL:" standing for process plan, equipment instruction, and cutter location, respectively. Next, the part name will be sent and then an acknowledgement of "READY" is expected from AMPLE. The text is then transferred followed by the number of bytes sent and then the checksum. After the checksum, the string "END_PP", "END_EI", OR "END_CL" is sent. To mark the end of a set of data, TWS sends an end-of-transmission character, i.e. 04H, to AMPLE. The "eot" character is acknowledged by a "READY". Finally, to end the entire transmission of data, TWS sends the command "COMPLETED" to AMPLE followed by a linefeed, i.e. 0AH.

After checking the data sent by TWS, the AMPLE system will send status messages on these data. It begins transmission with the commands "CHECKING" and "START". The "START" is acknowledged by "READY" and line feed character. If the data sent by TWS is correct, AMPLE will indicate this by sending a string "XX_CHECKOUT" where "XX" is replaced by "PP", "EI", or "CL". If there are errors in the data, AMPLE will send "XX_ERROR:" followed by the error description data. After receiving the error message, TWS stores it for later retrieval on the Winchester disc. To mark the end of the message, an "END_XX_STATUS" is sent and the next status message is expected. The string "COMPLETED" ends the session with TWS. Figure III.18 shows in graphic form the TWS/AMPLE communication protocol.

6. TWS CONTROLLER/DEVICE CONTROLLER COMMUNICATION PROTOCOL

The TWSC communicates with the device controllers through RS232C serial links. The communication is generally communicated in command/status pairs: the workstation controller sends a command to the device controller, and the device controller returns a status on the execution of the command. The commands sent to the device controllers are ASCII strings, with the content and the length of the strings differing from device to device. The status returned by the devices are ASCII strings as well. There are three different types of status: READY, BUSY, and ERROR. The actual characters sent by the device controllers are "r" for READY, "b" for BUSY, and "e" for ERROR. The error status usually contains an error code following the "e" to indicate what kind of error has occurred.



NOTE: Uppercase items are the actual strings sent. Lowercase items are the names of the items sent.

FIGURE III.18 TWS/AMPLE PROTOCOL

7. OPERATION SCENARIO

The following is a description of task execution during typical operation.

7.1 Stand-alone Mode

In the example below, the workstation is running in stand-alone mode. An operator keys in the type and the batch size of the part that is to be machined.

a) The operator orders a batch of parts through the user interface.

b) The task manager retrieves the proper process plan from the local database and decomposes it into a task level work element sheet and several device level work element sheets. The data sheets are written out to the local database.

c) The task manager invokes the task level controller. The task level controller is waiting at a semaphore for a data unit. When the task manager sends the data unit, the task level controller is ready to execute. However, it must wait until the task manager goes to sleep due to the task manager's higher priority.

d) The task level controller reads the work element sheet and device element sheets from the local database into RAM. It then executes the work elements by invoking the device level controller. After invoking the device level controller it goes to sleep.

e) The device level controller executes the device elements. It then invokes the task level controller and goes to sleep.

f) While the task and device level controllers are monitoring the manufacturing process, the task manager is idling and waiting for operator inputs. Once the operator enters a request at the keyboard, the task manager interrupts the task level and device level controllers and services the operator's request.

The above operation is illustrated in Figure III.19.

7.2 Integrated Mode

In the following example, the workstation is running in integrated mode. The AMRF cell controller sends batch information through the AMRF Network, and data exchange with AMRF database is made.

a) The task manager idles while waiting for command mailgrams from the cell controller. Meanwhile, the cell communication task scans the remote command mailbox once every 100 milliseconds for remote input.

1. Operator request reaches task manager.
2. Task manager invokes task level controller
3. Task level controller invokes device level controller.

NOTE: While the task and device level controllers are executing, the task manager may pre-empt them when an operator request arrives.

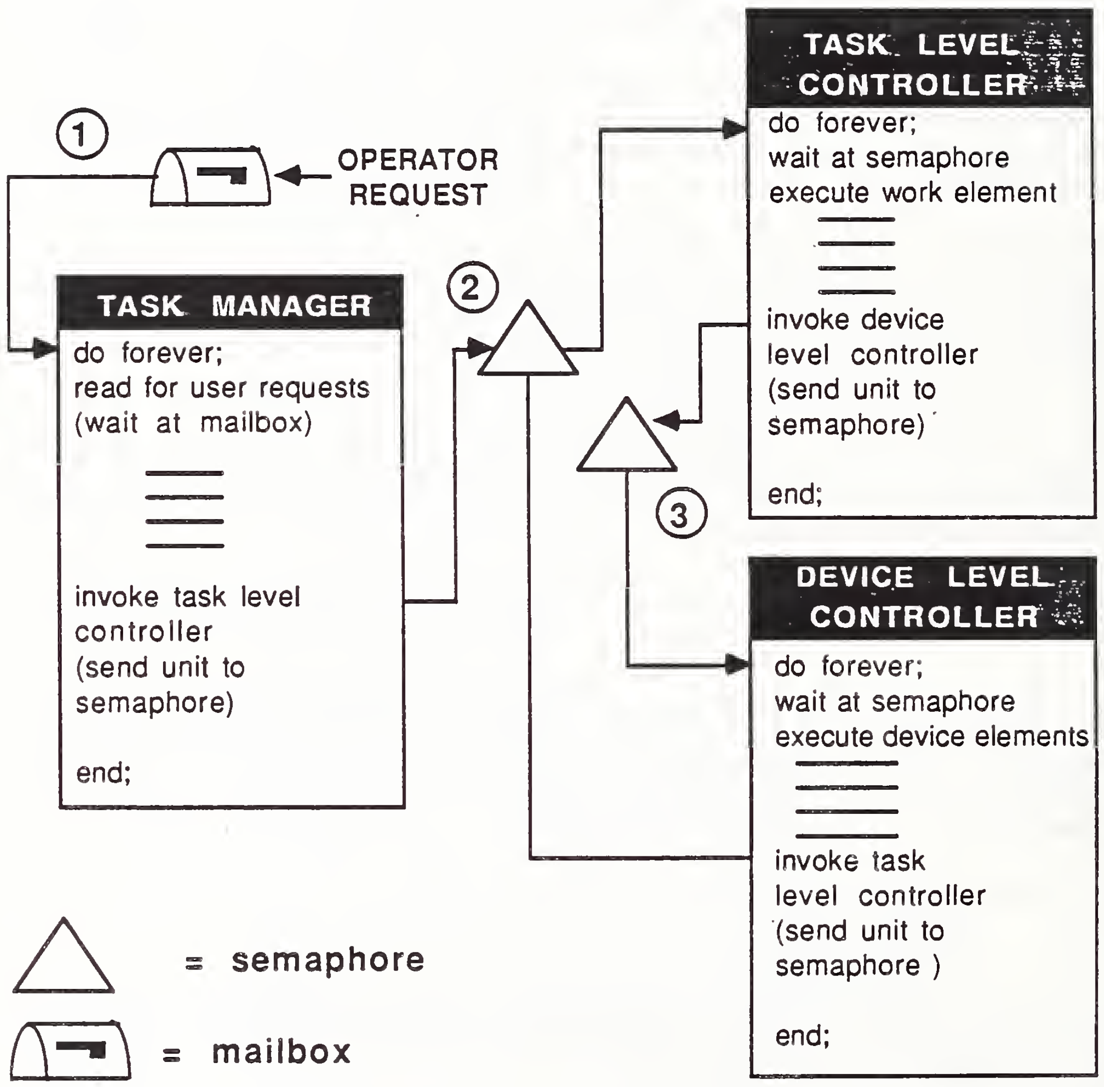


FIGURE III.19. TASK SYNCHRONIZATION BETWEEN JOB LEVEL CONTROLLER/TASK MANAGER, TASK LEVEL CONTROLLER, AND DEVICE LEVEL CONTROLLER

b) The cell controller sends the mailgram. The cell communication task deciphers the command and sends it to the task manager and wakes it up.

c) The task manager invokes the database communication task. The task manager passes the process plan id, which is found in the command mailgram from the cell controller, to the database communication task. After the communication task is invoked, the task manager goes to sleep to relinquish the CPU to the communication task.

d) The database communication task retrieves the process plan from the AMRF database. When this is completed, the task wakes up the task manager again and goes to sleep itself.

e) The task manager decomposes the process plan into task level and device level work element sheets.

The rest of the operation is the same as it is in stand-alone mode. The only difference is that the task manager updates the cell controller on the status of the batch.

8. FUTURE DEVELOPMENT

Future development for the TWS controller will include the development of a logging system that will record all the events that occur during operation to facilitate system trouble shooting. TWSC could also keep a tool wear profile for each tool used in the workstation so that an operator can replace a tool before it wears out. Another issue that may be addressed includes automating the generation of the workstation process needed to machine a particular part. This generation is currently accomplished by system programmers. Automating error recovery is another possible development.

IV. HIGH-LEVEL MACHINE TOOL CONTROLLER

The high-level machine tool controller is one level below the workstation controller in the Turning Workstation hierarchy. Its main function is to establish the necessary communication and control link between the workstation controller and the Computer Numerical Control (CNC) turning center and its auxiliary devices such as the interface module for the CNC keyboard, the collet changer controller, the tool-setting station controller, and the multichannel digital temperature measurement system. Another function of the high-level machine tool controller is the enhancement of the turning center accuracy by real-time error compensation for geometric and thermally induced machine tool errors. The details of the design and the operation of this controller are given in the following sections of this document.

1. DESIGN OF THE HIGH-LEVEL MACHINE TOOL CONTROLLER

1.1 Overview

The high-level machine tool controller is designed using off-the-shelf electronic components, such as a single-board microcomputer and communication boards. All the components are selected to be Multibus-compatible to create a modular system. Due to its modularity, this design allows the system builder to add additional boards, such as communication and/or data acquisition boards as the need arises.

The system software is written in a high-level language, PL/M. No operating system is used for the microcomputer. The software is designed to be structured and modular so that it can be maintained easily and modified for future needs.

1.2 Control Architecture

The high-level machine tool controller is a task-driven system. It receives tasks from the workstation controller, and decomposes these tasks into several subtasks for the lower-level controllers, such as the keyboard interface module, the collet changer, and the turning center CNC. By sending appropriate commands corresponding to these subtasks to the controllers of the auxiliary devices, the high-level machine tool controller coordinates the operations of these devices. After sending the commands to the lower-level controllers, it waits for the task completion status from the corresponding lower-level controller. When all the subtasks are completed, the high-level machine tool controller sends the "Ready" status signal to the workstation controller. When there is no new command from the workstation controller, the high-level machine tool controller goes into the idle state and waits for a new command. In order to carry out the cutting operation based on the NC part programs, the high-level machine tool controller accesses the AMRF database. When needed, it downloads necessary NC part programs from the database to the CNC controller of the turning center to activate them for cutting operation. It also has the capability of uploading any NC part programs from the CNC controller of the turning center to the database.

The communications between the high-level machine tool controller and all the other lower level controllers are through serial RS232 ports. On the other hand, communications between the high-level machine tool controller and the workstation controller are either through a serial RS232 interface or through the AMRF network. The protocols for these two modes of communications are totally different. Strings of ASCII characters are used for command and status information in the network communication mode. In the serial communication mode, a single ASCII character is used for each command and status between the high-level machine tool controller and the workstation controller.

In the error compensation mode, the geometric and thermally induced machine tool errors are predicted as a function of machine axis position, direction of motion, and the machine tool temperature profile based on the previous calibration measurements. During machining, these errors are calculated, converted into servo counts and injected into the "following error" registers of the CNC controller in real-time. In this mode, the high-level machine tool controller runs in synchronization with the CNC controller. Thus, position information is updated and error correction is injected every servo control cycle of 20 milliseconds. The whole error compensation operation is transparent to the user of the system.

1.3 Hardware Components

The high-level machine tool controller is a multibus single-board microcomputer (Intel iSBC 86/30) with 128k RAM and 64k EPROM memory. This microcomputer board contains a 16-bit 8086 microprocessor as the CPU, and a high-speed version 8087A numeric coprocessor for floating point arithmetic operations. The architecture of this board is designed for high-speed floating point numeric computations which are necessary for the real-time error compensation function of the high-level machine tool controller. The combination of 8086 and 8087A makes it possible to run the computer at an 8-MHz clock rate to meet the requirement of high servo bandwidth for error compensation during contouring cuts. This microcomputer uses two Multibus serial I/O boards, with four RS232 serial I/O ports on each, to communicate with the controllers of the auxiliary devices mentioned before. The communications between the CNC controller of the turning center and the high-level machine tool controller are done through three multibus parallel I/O boards.

In addition to the main microcomputer unit, the high-level machine tool controller uses two other auxiliary modules to carry out its function. One of these two modules is the CNC keyboard interface module. The keyboard interface module is designed and built in house, and based on an 8048 single component microcomputer. The function of this module is to translate commands from the high-level machine tool controller and enter them into the CNC controller of the turning center by emulating the keyboard operation of the turning center CNC. This module is necessary for the high-level machine tool controller to be able to send MDI (manual data input) commands to the CNC for upload/download NC part programs, activate them, and execute single-block axis motions.

The other auxiliary module is the digital temperature measurement system which is used to monitor the temperatures around the machine structure. This temperature information is used in the error compensation scheme employed by the high-level machine tool controller. The temperature measurement system consists of signal conditioning and digitizing electronics, and a 10-channel scanner. It communicates with the high-level machine tool controller through an RS232 interface. Upon receiving a command from the high-level machine tool controller, the temperature measurement system digitizes the appropriate channels and returns the temperature information.

1.4 Software Components

The high-level machine tool controller software is written in a high-level language, PL/M. The main criteria in designing the system software are flexibility, modularity, and easy maintainability. Selection of a high-level language helps to meet these criteria. The system software consists of four modules: the main module, a module for database operations, a module for error compensation, and a module for parallel communications between the high-level machine tool controller and the turning center CNC. Each module has a series of procedures which are called from the main program in the main module. Brief descriptions of these procedures are given below.

1.4.1 Main Module

This module includes the main program of the high-level machine tool controller. In addition, it has the following procedures:

KB_CMD: This procedure is used to communicate with the keyboard interface module.

AB_CMD: This procedure is used to send commands to the CNC through its parallel port.

UPLOAD_NC_PRG: This procedure is used to upload the NC part programs from CNC to the local memory.

UPLOAD_LCL: This procedure is used to upload the NC part program from the local memory to the local database.

DOWNLOAD_NC_PRG: This procedure is used to download the NC part program from the local memory to the turning center CNC.

READ_CMD_LCL: This procedure is used to receive the commands from the workstation controller in the local mode.

ACTV_PRG: This procedure is used to activate the NC part program in the CNC, after it is downloaded to the CNC, in preparation for running the NC program.

PRG_CANCL: This procedure is used to cancel the NC part program in the CNC memory to eliminate error conditions when it is downloaded the next time.

COMP_STR_LCL: This procedure is used to interpret the workstation commands received locally.

SIMULATE_EXEC: This procedure is used to simulate the execution of the tasks which are assigned by the workstation controller. This is necessary for the test mode of the operations.

EXEC_COMMAND: This procedure is used to actually execute the tasks assigned by the workstation controller.

ERR_REC: This procedure is used for the limited recovery of error conditions, e.g., missing database or lower-level controller statuses.

1.4.2 DBASE Module

This module consists of procedures, which are used in the remote operation, for accessing the AMRF database or communicating with the workstation controller using the AMRF network. Brief descriptions of these procedures are given in the following.

INIT_DB: This procedure is used to initialize the database connection.

INT$TO$STR: This procedure is used to convert an integer variable into a string variable to be able to display it to the screen.

START_DB: This procedure is used to start the database to prepare for operations.

SHUT_DB: This procedure sends the necessary command string to the database to terminate the operations.

SET_CMD_DB: This procedure prepares the command string to be sent to the database.

WRIT_CMD_DB: This procedure is used to write the command string created with the previous procedure into the AMRF network mailbox to be transmitted to the database.

DISP_CMD_DB: This procedure displays the command sent to the database on the CRT screen.

READ_STA_DB: This procedure reads the database status mailbox via the AMRF network and stores it into the local memory.

DISP_STA_DB: This procedure displays the recently read database status.

CHECK_STA_DB: This procedure is used to interpret the database status string, which is copied from the network mailbox to the local memory.

READ_DAT_DB: This procedure is used to read the data mailbox to receive the data sent by the database. Once the data is read, it is stored in the local memory.

DISP_DAT_DB: This procedure is used to display the data received from the database on the CRT screen.

WRIT_DAT_DB: This procedure is used to send the data to the database when uploading NC part programs. It writes the data to the data mailbox of the AMRF network.

UPLOAD_DB: This procedure is used to upload the NC part program to the database. After preparing the NC program for uploading, it calls the WRIT_DAT_DB procedure to copy it into the data mailbox.

DOWNLOAD_PRG_DB: This procedure is used to download the NC part program from the database to the local memory.

MC_STATUS: This procedure creates the high-level machine tool controller status string to be sent to the workstation controller.

DISP_STA: This procedure is used to display the machine tool controller's status, which is created by the previous procedure.

WRIT_STA: This procedure is used to copy the previously created machine tool controller's status to the status mailbox of the high-level machine tool controller.

READ_CMD: This procedure checks if there is a command in the workstation command mailbox; if there is, it copies it into the local memory.

DISP_CMD: This procedure is used to display the recently received workstation controller command on the CRT screen.

COMP_STR: This procedure is used to interpret the command received from the workstation controller.

1.4.3 ERROR Module

This module consists of the procedures which calculate the error components of the overall geometric and thermally induced errors, with the total resultant error components in x and z directions to be injected into the turning center servo loop. Brief descriptions of these procedures are given below.

HDNG$SAFE, HDNG$SAFE2, HDNG$MOVE$GAGE, and HDNG$MOVE$INC procedures are used to move the turning center cutting tool to prepare for the tool setting operation.

ARM_DOWN: This procedure is used to bring the tool setting station arm down for a tool setting operation.

TSS_READ: This procedure is used to communicate with the tool setting station controller to read the displacement readings of the LVDT of the tool setting station.

ARM_UP: This procedure is used to bring the tool setting station arm up upon completion of the tool setting operation.

MC_OFFSETS: This procedure is used to calculate the machine reference position drift using the tool setting station data.

TEMP_SCAN and TEMP_READ: These procedures are used to scan and read the various temperatures around the machine structure.

SPYAW, ORTHOG, ZYAWT, DSPX, EPSX, XDISP, DSPZ, EPSZ, ZDISP, XYAW, ZYAW, Z$STR$X, AND X$STR$Z procedures are used to calculate the individual error components as functions of temperature and position.

ERR_CALC: This procedure is used to combine the error components to calculate the resultant error in x and z directions.

ERROR_CORRECT: This procedure converts the calculated resultant error components into machine tool servo counts and calls the procedure to inject into the machine servo loop.

INIT_ERR: This procedure initializes the coefficients used in other procedures.

1.4.4 PALCOM Module

This module consists of procedures, which are used for communication between the high-level machine tool controller and the CNC of the turning center. These communications are through parallel I/O ports and transfer position and correction information back and forth.

READ_POS: This procedure is used to read the position of the axes of the turning center.

OUT_CORR: This procedure is used to output the correction values to the CNC.

OUT_CORR_X14: This procedure is used to send a special code to the CNC indicating the high-level machine tool controller's desire to initiate the error correction scheme.

OUT_CORR_0: This procedure is used to send a special code to the CNC to stop the communications for error correction.

2. OPERATION OF THE HIGH-LEVEL MACHINE TOOL CONTROLLER

Currently, there are six operation modes of the high-level machine tool controller. Three of these modes are used mainly for communication debugging purposes. These are: 1) a communication test between the workstation controller and the high-level machine tool controller; 2) a communication test between the workstation controller, the machine tool controller, the AMRF database, and the CNC; and 3) a test of local communication between the workstation controller and the high-level machine tool controller. In test modes there is no real execution of these tasks. All the commands received are interpreted but not executed. Status information is returned after the interpretations are displayed on the CRT screen. The other two modes are local and remote operation modes. Upon power up, the system goes through its own initializations and then comes up with the operation menu. After the selection is made, all the operator has to do is answer some preparatory questions. Then, the system runs on its own without any intervention. Since in the communication test modes no operator intervention is required, only the last two modes are described below.

2.1 Local Operation Mode

In local mode, the high-level machine tool controller communicates with the workstation controller through a serial I/O port. In this mode the database for the NC part programs and tool offset data tables is the local database residing in the workstation controller system. The command and status information between the workstation controller and the high-level machine tool controller is passed from one to the other as single ASCII characters. The following list is a sample group of commands from the workstation controller:

g: loosen collet
h: tighten collet
f: send w axis to home position
k: send x and z axes to home position
l: run the NC part program to make the part

After receiving each command from the workstation controller, the high-level machine tool controller compares it with the set of acceptable commands stored in the memory, if the command is acceptable then it starts executing accordingly and sends a "Busy" status back to the workstation controller. When the task is completed, it sends a "Ready" status to the workstation controller and starts waiting for a new command.

2.2 Remote Operation Mode

In the remote operation mode, the communications between the high-level machine tool controller and the workstation controller are through the AMRF network. The AMRF database is used in this mode instead of the local database. These communications use protocols similar to those employed in the other stations of the AMRF. Different command and status mailboxes, which are allocated in the common memory, are used in these transactions. Based on the sequence numbers, these mailboxes are transferred by the network server. In this mode, instead of single characters, complete texts are used for each command. Some examples are given in the following:

COLLET LOOSEN	: loosen the collet
COLLET TIGHTEN	: tighten the collet
W HOME	: send w axis to home position
XZ HOME	: send x and z axes to home position
MAKE PART	: run the NC part program to make the part

In this mode, upon power up, the operator has to answer some additional questions such as if he wants to select NC program list from the database, or upload/download NC program to/from the database, or reset the mailbox sequence numbers to zero. After the operator goes through this self-explanatory question-answer period, the real operation of the high-level machine tool controller does not differ from the local operation mode. During the operation, an "R" from the keyboard will reset the controller, and a "Q" will show the status information from the current operation. During this inquiry, it is possible to change the status of some of the devices listed on the screen to prevent the controller from getting caught up in an indefinite status request mode while the controller on the other end can not respond due to a malfunction situation.

3. SUGGESTIONS FOR FUTURE DEVELOPMENT

In order to decrease the number of controllers in the system, some of the lower level controllers such as the tool setting station controller, or the keyboard interface controller can be incorporated into the high-level machine tool controller. The high-level machine tool controller computer has enough capability to carry out the functions of these controllers especially if the operations of these controllers are done sequentially rather than parallel to the functions of the high-level machine tool controller. The trade-off of this situation is that the software for such a combined controller would be more complex and difficult to debug and maintain.

V. ROBOT CONTROLLER INTERFACE

The Robot Controller Interface (RCI) is a microprocessor-controlled device which allows the workstation controller to communicate with both the robot controller and the tray station controller by means of a serial communication line. The robot controller, a Bendix Dynapath System 5A CNC (Computer Numerical Control), was designed primarily for manual operation. Provisions were added for external control, but without an interface, the Turning Workstation controller would need to control and monitor many separate lines in order to control the robot. The RCI provides the communication between the Turning Workstation controller and the tray station controller by "passing" commands from the workstation controller to the tray station controller, and acknowledgement from the tray station controller to the workstation controller.

1. INTERFACE DESIGN

1.1 Overview

The RCI was designed to simplify communication between the workstation controller and the robot controller. Previously, to control the execution of individual robot programs, the workstation controller needed to control or monitor more than a dozen signal lines. This job is now done by the RCI, which communicates with the workstation controller via a serial (RS232) communication line (see Figure V.1). When the workstation controller calls for a particular robot program, the interface takes care of actually calling it up from the robot controller, freeing the workstation controller of this low-level task. After the robot program has completed execution, the interface notifies the workstation controller and waits for the next command.

The RCI performs three main functions. The first is to emulate the keyboard and switches on the front panel of the robot controller. These are normally used by an operator to call and execute robot programs. The second is to monitor the status of the end-of-program line from the robot controller, in order to know when a program has completed execution. The third function is to provide an interface between the workstation controller and the tray station controller. Before a batch of parts is started, the workstation controller commands a tray to be held in place, and releases it to the robot cart when the batch is completed.

1.2 Control Architecture

The control consists of initiating the robot tasks called for by the workstation controller and periodically (every 50 milliseconds) reading control lines from the robot controller and several front panel switches. The RCI detects completion of a robot task and notifies the workstation controller that the robot is ready for its next task.

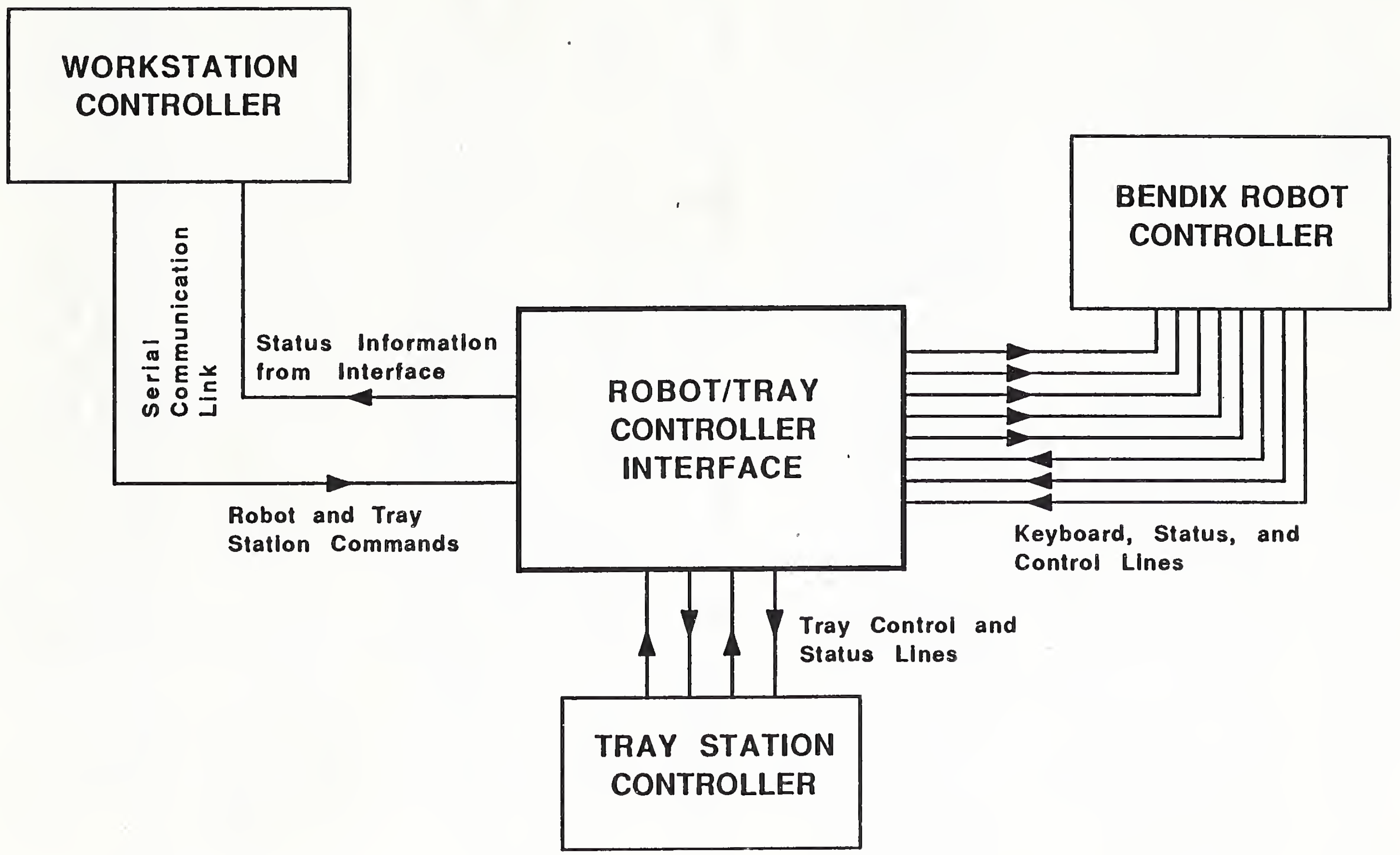

WORKSTATION CONTROLLER
Serial Communication Link
Status Information from Interface
Robot and Tray Station Commands
ROBOT/TRAY CONTROLLER INTERFACE
BENDIX ROBOT CONTROLLER
Keyboard, Status, and Control Lines
Tray Control and Status Lines
TRAY STATION CONTROLLER

FIGURE V.1. BLOCK DIAGRAM OF THE ROBOT/TRAY CONTROLLER INTERFACE CONFIGURATION

A robot task is initiated by calling the appropriate robot program. The RCI does this by emulating the robot controller keyboard, using reed relays in place of the robot controller's pushbuttons. To detect completion of a robot task, the RCI monitors the appropriate robot controller status signal line.

There is a difference between total robot control and control of the execution of robot programs. The RCI does not control the movement of the robot or cause the robot controller to move it in any way other than under control of a previously entered robot program. Other than that, the only influence the RCI has over the movement of the robot is to inhibit all motion by activating the robot controller's Interpolator Enable line.

1.3 Electronic Components

The electronic components in the RCI consist of an 8-bit microprocessor (Intel 8031), an Erasable Programmable Read Only Memory (EPROM) with 64K bytes of memory storage, a number of 8-bit latches, reed relays and opto-isolators, along with various other support components. The 8031 microprocessor has an 8-bit data line, bidirectional ports, timer/counters, and external interrupt lines. There are also 128 bytes of volatile memory located on-chip.

Some considerations which went into the choice of a microprocessor for this job are :

- The availability of development tools to debug the initial prototype.
- Is the microprocessor supported by a high-level language? This is not necessary, but it does make the writing, debugging, and documentation of the software easier.
- Are the I/O (Input/Output) ports easy to control and manipulate?
- What kind of communication is intended between the microprocessor and (in this case) the workstation controller? Can the microprocessor accomplish this kind of communication?
- Are there enough I/O lines, counter/timers, and interrupts to do the job?

The Intel 8051 family of microprocessors is supported by PLM-51, which is a high-level language, as well as development tools for developing, testing, and debugging the design.

The I/O lines have byte or bit control capability, making them easy to use for this design. A serial communication port is included in the microprocessor making serial (RS232) communication with the workstation controller an easy task. No external interrupt lines were needed, but two timers were necessary - one to set the serial communication baud rate, and the other to set the cycle time for the updating and sampling routine.

There were not enough I/O lines to handle the large number of input and control signals, so 8-bit latches (74LS373) were used and multiplexed onto the 8-bit data bus of the microprocessor.

The RCI program is contained in approximately 3 kilobytes of memory and is stored in the 64K EPROM. The 8-bit latches (74LS373) are used to latch incoming information for the microprocessor to read and to latch outgoing information from the microprocessor to the robot controller. The reed relays are used to simulate the pushbuttons, rotary switches, and toggle switches on the robot controller. Electrically, the relays are in parallel with the robot controller switch contacts. The opto-isolators are used to translate the voltage levels between the RCI (5-volt logic) and robot controller (24-volt logic), as well as for protection against damaging voltage spikes on either side.

The implementation of the RCI to the robot controller is conceptually straightforward. The switches on the robot controller, including the keyboard, toggle, rotary and other pushbutton switches, are grouped together with eight or less to a group. Each group is sampled by the robot controller in turn - that is, each group has its own time slot within which it is monitored for any change in switch status. The total sample time is approximately 50 milliseconds.

When a switch is manually activated, the robot controller infers which group the switch belongs to by knowing the time slot that the active signal occurred in. The particular switch is then known as a specific one of eight within that group.

Ideally, an interface device would monitor the robot controller as closely as possible to its (the robot controller's) microprocessor. If it was possible to "tap in" to the robot controller's data and control bus, control could be done entirely electronically. Another benefit would have been very reliable checking, due to feedback on the data and control lines, of the currently requested robot programs. Unfortunately, this approach did not prove feasible due to a lack of robot controller software documentation.

The alternative decided upon was an electromechanical (reed relay) interface which mimics the robot controller switches. This method is implemented by severing the appropriate switch sampling lines. The relays are inserted in a fashion which provides two main modes of operation. The default is the transparent mode. The reed relays are configured such that in their normal state the robot controller operates in its usual, manual mode. The RCI is transparent - it does not affect the operation of the robot controller. In the second mode, the robot controller switches are disabled, and the RCI reed relays emulate the appropriate switches and controls, thus controlling the signals which the robot controller receives internally.

1.4 Software

Software for the RCI is written in PLM-51, a high-level language for the Intel 8051 family of microprocessors. The program is linked with the PLM-51 library, compiled and stored in an EPROM, occupying approximately 3000 bytes of code.

PLM is a structured language, and the program is written mainly as a series of procedures. These procedures define the commands, relay combinations, time lengths for contact closure, status and error messages and the serial communication scheme. The main program loop ties everything together by checking flags and calling the appropriate procedures. The software also keeps track of the mode of operation, selected by switches on the front panel of the RCI. Whenever the RCI is in control of the robot, the robot is disabled, unless explicitly enabled by a move command from the workstation controller. This is designed to prevent the robot from moving unexpectedly for any reason.

There are two interrupt procedures. Both are called by internally generated interrupts. One interrupt is generated when a data flow is detected into or out of the microprocessor's serial port register. This occurs when the RCI is either sending or receiving serial communication. The other interrupt is generated when a timer (within the microprocessor) overflows. This timer is preloaded with a value which causes it to overflow every 5 milliseconds. When this overflow interrupt occurs, it causes program execution to exit the main loop and begin the timer interrupt procedure. This procedure directs the microprocessor to take readings of the various inputs to the RCI, and update he output relays and program variables. At the end of the interrupt ocedure, program execution jumps back to the main loop and continues where it left off, but with updated information.

Generally, the process begins at power-up (or reset) with a setup procedure which disables robot motion, initializes all registers and control variables, and configures all of the relays and output lines so that the RCI seems transparent to the robot controller. This includes loading the internal baud rate timer and the 5 millisecond timer (above) with the proper values, setting the interrupt priorities in case they both occur at the same time, configuring the serial communication register, and finally, starting the timer and enabling the interrupts.

When the setup routine is completed, the program begins executing the repetitious main loop, jumping out temporarily because of a timer overflow interrupt (every 5 milliseconds) or a serial communication interrupt.

Whenever the workstation controller sends a command to the RCI, the characters of the command are stored, character by character as they are received, in a buffer area of memory. Each time a character (i.e., a letter or a control character) is received by the microprocessor's serial communication register, an interrupt ("serial interrupt") is generated. When this happens, execution temporarily jumps from the main loop to the serial interrupt procedure. Here, they are placed, as they are received, into a buffer area until the complete command is assembled. Since the characters are being transmitted at a rate of

1200 baud, there is plenty of time for the microprocessor to return execution to the main loop until the next character is received and generates another interrupt. The incoming command is complete when a carriage-return character ("CR") has been received. The interrupt procedure places the complete command in a command buffer and sets a "command pending" flag, to be read in the main loop. After execution jumps back to the main loop and this flag is found to be set, a procedure is called to determine which command it is and how to respond to it. If it is an unrecognized command, an error message is generated, and sent to the workstation controller. This command-handling procedure also determines what numeric parameter, if any, was sent with the command.

Each command has its own sub-procedure. The current command (including numeric parameters, if present) is matched with the correct sub-procedure, or command procedure. These command procedures first call another procedure to send an acknowledgement character back to the workstation controller. This acknowledges that the command has been received. Another character will be sent (later) upon completion of the action called for by the workstation controller. The procedure then checks for out of limit numeric parameters, and sends an error message if necessary. If there are no error conditions, the command procedure then calls other procedures to complete the action, then jumps back to the main loop.

2. OPERATION

There are three modes of operation, as described below. The local mode is used primarily for test and diagnostic purposes. A terminal, instead of the workstation controller, is used to enter commands which change parameters and monitor status, as well as control robot program execution. The remote mode is used during automated operation with the workstation controller. In this mode there are normally only a few different commands sent from the workstation controller to the RCI. When either the remote or local mode is selected, robot motion is automatically inhibited unless explicitly commanded otherwise. The transparent mode is used when the robot controller is being operated manually, as when "teaching" new positions to the robot. In this mode, the RCI is completely transparent to the robot controller, and robot motion is not inhibited.

2.1 Local Operation

2.1.1 Setup Requirements

An RS232 terminal is used for local control and operation of the RCI. Incoming data is received by the RCI on line 2, and data is sent to the terminal over line 3. Line 7 is used as the signal ground, as is standard. The terminal should be set to full duplex mode, no parity checking, and with a baud rate of 1200. In local mode, the characters are echoed back to the terminal, and a carriage return/line feed response is included.

2.1.2 Commands and Responses

The RCI commands are listed below. The ENTER (or carriage return) key must be pressed to complete each command, as the software looks at it before setting the "command pending" flag in the program loop. If the command includes numeric parameters, they are indicated within < >. The standard response to any command from the workstation is a "b" (for "busy") upon receipt of the command, and an "r" (for "ready") after completion of the command. For the two tray station controller commands, however, the responses are "bt" and "rt", respectively. This is to distinguish the tray station status from the robot status. In local mode these character responses are each followed by a line feed (LF) and a carriage return (CR).

In the following discussion, the quote marks shown are not included with the command, nor are the blank spaces. For example, "b LF CR" is actually sent bLFCR, and "td CR" is entered tdCR.

cr CR Control Reset

This command activates the Control Reset switch on the robot controller. Control Reset clears the robot controller of the current robot program, and is activated prior to a new program call. The response to this command is a "b LF CR" upon receipt and a "r LF CR" after completion.

cs CR Cycle Start

This command activates the Cycle Start switch of the robot controller. Cycle start is used, after a robot program is called to begin execution. It is also used to resume execution during the program if a Motion Hold has been activated. The response to this command is a "b LF CR" upon receipt and a "r LF CR" after completion.

ec <0 or 1> CR Echo

This command sets or resets the ECHO flag. When the ECHO flag is set (1), as it would be for local operation, the command sent to the programmable stop is echoed back to the terminal screen, with a LF (line feed) character. When reset (0), there is no echo or LF. The response to this command is a "b LF CR" upon receipt and a "r LF CR" after completion.

lm <0 or 1> CR Latch Monitor

This command is used as a diagnostic in the Local Mode to monitor the state of the input latch lines. When activated (with a "1") a binary string is displayed on the terminal, indicating the state of each individual input line. The binary string will change to reflect input line changes. The response to this command is a "b LF CR" upon receipt and a "r LF CR" after completion.

mh CR Motion Hold

This command activates the Motion Hold switch of the robot controller. Motion Hold is a temporary halt of robot motion and program execution. Both are resumed with the activation of Cycle Start. The response to this command is a "b LF CR" upon receipt and a "r LF CR" after completion.

rt <1 through 999> CR Robot Program Call

When this command is received, a series of actions is performed, resulting in a robot program being called and its execution begun. A robot controller "end of program" signal is monitored by the RCI. When this signal is detected, robot motion is inhibited. The response to this command is a "b LF CR" upon receipt and a "r LF CR" after completion.

sg <0 through 35> CR Signal lines

This command is used as a diagnostic in the Local Mode to test the individual output lines. The specified output signal is pulsed. The response to this command is a "b LF CR" upon receipt and a "r LF CR" after completion.

st CR Start

This command is a software reset and is used to initialize all registers and inhibit robot movement. The response to this command is a "r LF CR" after completion.

tr <1 or 2> CR   Tray Request

This command activates the Request line for Tray 1 or Tray 2. This is done prior to starting a batch of parts in the turning center. The robot cart is thus prevented from changing or moving the tray, as the robot will be picking up blanks and dropping off finished parts to it. The response to this command is a "b LF CR" upon receipt and a "r LF CR" after completion.

td CR   Tray Deactivate

This command frees the tray from control of the turning center, and allows access to it by the robot cart. The command is typically sent after the turning center has completed a trayful of parts. This command is not specific to the tray number, though it can be made so. As is, it deactivates all trays. The response to this command is a "b LF CR" upon receipt and a "r LF CR" after completion.

2.2 Remote Operation

2.2.1 Setup Requirements

Remote operation entails communication with, and control by, the workstation controller. The setup requirements are similar to those for local operation (above). The baud rate is set at 1200, and a standard 25-pin D-subminiature (RS232) connector is used. Data to the workstation controller is transmitted on line 3, data is received on line 2 and line 7 is signal ground. Remote operation differs from local operation in that there is no character echo back to the workstation controller, and the LF character is not included in the responses.

2.2.2 Commands and Responses

The RCI commands used in remote operation, listed below, are a subset of the commands available for use in local mode. Again, the CR character must follow the command, and the spaces and quotes are not included. The responses, "b" and "r", are the same also, except the LF character is not included. For the two tray station controller commands, however, the responses are "bt" and "rt", respectively. This is to distinguish the tray station status from the robot status.

cr CR   Control Reset

This command activates the Control Reset switch on the robot controller. Control Reset clears the robot controller of the current robot program, and is activated prior to a new program call. The response to this command is a "b CR" upon receipt and a "r CR" after completion.

cs CR   Cycle Start

This command activates the Cycle Start switch of the robot controller. Cycle start is used, after a robot program is called to begin execution. It is also used to resume execution during the program if a Motion Hold has been activated. The response to this command is a "b CR" upon receipt and a "r CR" after completion.

mh CR   Motion Hold

This command activates the Motion Hold switch of the robot controller. Motion Hold is a temporary halt of robot motion and program execution. Both are resumed with the activation of Cycle Start. The response to this command is a "b CR" upon receipt and a "r CR" after completion.

rt <1 through 999> CR   Robot Program Call

When this command is received, a series of actions is performed, resulting in a robot program being called and its execution begun. A robot controller "end of program" signal is monitored by the RCI. When this signal is detected, robot motion is inhibited. The response to this command is a "b CR" upon receipt and a "r CR" after completion.

st CR Start

This command is a software reset and is used to initialize all registers and inhibit robot movement. The response to this command is a "b CR" upon receipt and a "r CR" after completion.

td CR Tray Deactivate

This command frees the tray from control of the turning center, and allows access to it by the robot cart. The command is typically sent after the turning center has completed a trayful of parts. This command is not specific to the tray number, though it can be made so. As is, it deactivates all trays. The response to this command is a "bt CR" upon receipt and a "rt CR" after completion.

tr <1 or 2> CR Tray Request

This command activates the Request line for Tray 1 or Tray 2. This is done prior to starting a batch of parts in the turning center. The robot cart is thus prevented from changing or moving the tray, as the robot will be picking up blanks and dropping off finished parts to it. The response to this command is a "bt CR" upon receipt and a "rt CR" after completion.

2.3 <u>Transparent Mode Operation</u>

2.3.1 Setup Requirements

When the RCI is in transparent mode it has no effect on the operation of the robot controller. A switch on the front panel of the RCI selects this mode of operation. When selected, commands sent to the RCI either locally (a terminal) or via the workstation controller are recognized. This mode does not automatically inhibit the motion of the robot, as do the local and remote modes, above.

3. SUGGESTIONS FOR FUTURE DEVELOPMENT

One drawback of the switch emulation approach to control is the lack of feedback. When the RCI receives a command from the workstation controller to begin execution of a certain robot program, the correct switches are activated via relays. Aside from visual confirmation (the robot controller CRT screen), there is no way of knowing in advance which program the robot is going to execute.

In order to know this, the robot controller's data and control lines would have to be monitored. Any accessible feedback would, at best, confirm that the robot controller received the intended signals, but not if the robot was about to execute the correct program.

The idea devised to work around this problem was to prefix each robot program with a unique binary number. When program execution began, the number would be decoded and the execution halted if it was not correct. This was not implemented, however, due to time constraints.

Another improvement to the RCI could be a real-time clock connected to the data bus. This would enable the RCI to list or keep track of many possible events. Failures, down time, or even a simple log of all transactions for a certain period of time would be a valuable diagnostic tool.

VI. ROBOT GRIPPER

In general, end effectors are designed with a specific task in mind and are consequently inflexible in use. To compensate for this, mechanisms have been designed to allow the robot to exchange one end effector for another. However, a large inventory of specialized end effectors can become quite expensive. At the Turning Workstation a single end effector, or gripper, was developed to handle a variety of tasks. This flexibility is achieved by changing individual gripping components, or fingers, rather than the entire gripper, which results in a much simpler and cost-effective operation. The gripper also utilizes two jaws instead of one, in order to minimize the transferring time. Other features of the design include a high grip-force-to-weight ratio and a high structural rigidity. The purpose of the gripper is to hold tools, collets, and workpieces while the robot transfers them to and from the machine tool.

1. DESIGN OF THE ROBOT GRIPPER

1.1 Overview

The gripper was designed to satisfy several requirements without exceeding certain physical constraints. The main constraint is due to the robot specifications on payload. The maximum payload allowable by the robot is 50 pounds and yet the gripper is required to hold objects up to 30 pounds. Therefore, the gripper was designed with high power-to-weight ratio air motors and light-weight aluminum structural components. One of the major requirements is for the gripper to handle workpieces in a variety of shapes and sizes. To achieve this flexibility each gripping finger can be easily removed and exchanged for a finger of different shape and capacity. In addition, the gripper is required to hold up to two workpieces at once, and be able to remove a workpiece from the collet and insert the opposite end without any refixturing. These latter two requirements are designed to minimize transfer time and maximize machine utilization. Because the workpieces that are removed and reinserted may extend from the collet by as little as 0.5 inch, it is necessary for the gripper to have a very narrow profile.

During tool changing, the gripper cannot be fully opened because of the space constraint at the tool turret. Therefore, the position of each finger must be controllable throughout the entire range of travel, which was accomplished through the use of encoder feedback. Also, due to very small clearances between the workpiece and the collet, it is necessary for the center of the gripping action to be repeatable to within 0.002 inch.

The gripper controller is a microcomputer-based system designed to control the operation of each jaw independently. The 8751 microcomputer has built-in random-access memory (RAM) for temporary data manipulation and erasable electrically-programmable memory (EPROM) for system program storage. The 8751 was chosen because it is best suited for development application requiring field updates. The microcomputer actuates the pneumatic motor to control the opening and closing of the gripper, while it closes the servo loop by reading the encoder for position feedback. The power supply and circuit board, containing the microcomputer and associated electronics, are contained in a chassis installed in the main control rack. The gripper controller communicates with the workstation controller or a higher-level controller through an RS232C serial port.

Since the controller was designed for the sole purpose of controlling the gripper, a massive operating system was deemed unnecessary. The software for the controller was written in PL/M, a highly structured programming language for microcomputers. The software is written in a modular fashion for ease of expansion.

1.2 Control Architecture

The gripper control is operated in a real-time mode; that is, the gripper controller responds to commands in the form of interrupts. When the controller receives a command through the serial or parallel interface, it executes a particular software procedure to accomplish the task. Once the task is completed, a status (ready) signal is returned to the higher level controller. An error signal and an error code are returned if a faulty condition exists. Commands from the higher-level controller are interpreted and executed one at a time. While one command is processed, additional commands are stored in a queue. These queued commands are served on a first-in-first-out (FIFO) basis. The gripper jaw can be opened completely, closed completely, or positioned to a certain commanded position. The maximum opening of the jaw is determined by proximity sensors and the jaw position feedback is determined by counting encoder pulses.

1.3 Mechanical Components

The robot gripper consists of three principal aluminum structural components: the frame, the hands, and the fingers. The frame consists of two tubes connected longitudinally with a center web as shown in Figure VI.1. Each hand is made to straddle a tube and uses the outer surface of the tube as a rail to slide on. A sleeve bearing is mounted between the hand and tube to prevent scoring of the tube surface and to reduce friction. Four hands are used altogether and are paired to form a single jaw for each tube. The fingers are attached to the hands and are the only components that actually come into contact with the workpiece. The term "workpiece" here refers to blank or machined parts, turret tools, or spindle tools.

Since two sets of fingers alone would not be able to handle such a large variety of workpieces, it was necessary to develop a method for the automatic exchange of fingers. Each finger mates with a hand via a dovetail slide. The female section of the dovetail is machined into the hand parallel to the tube axis. This type of configuration is important for two reasons. First, the wedge shape of the dovetail forces a pair of fingers to remain co-planar as a load is applied. Thus, the plane of the gripping action remains constant while

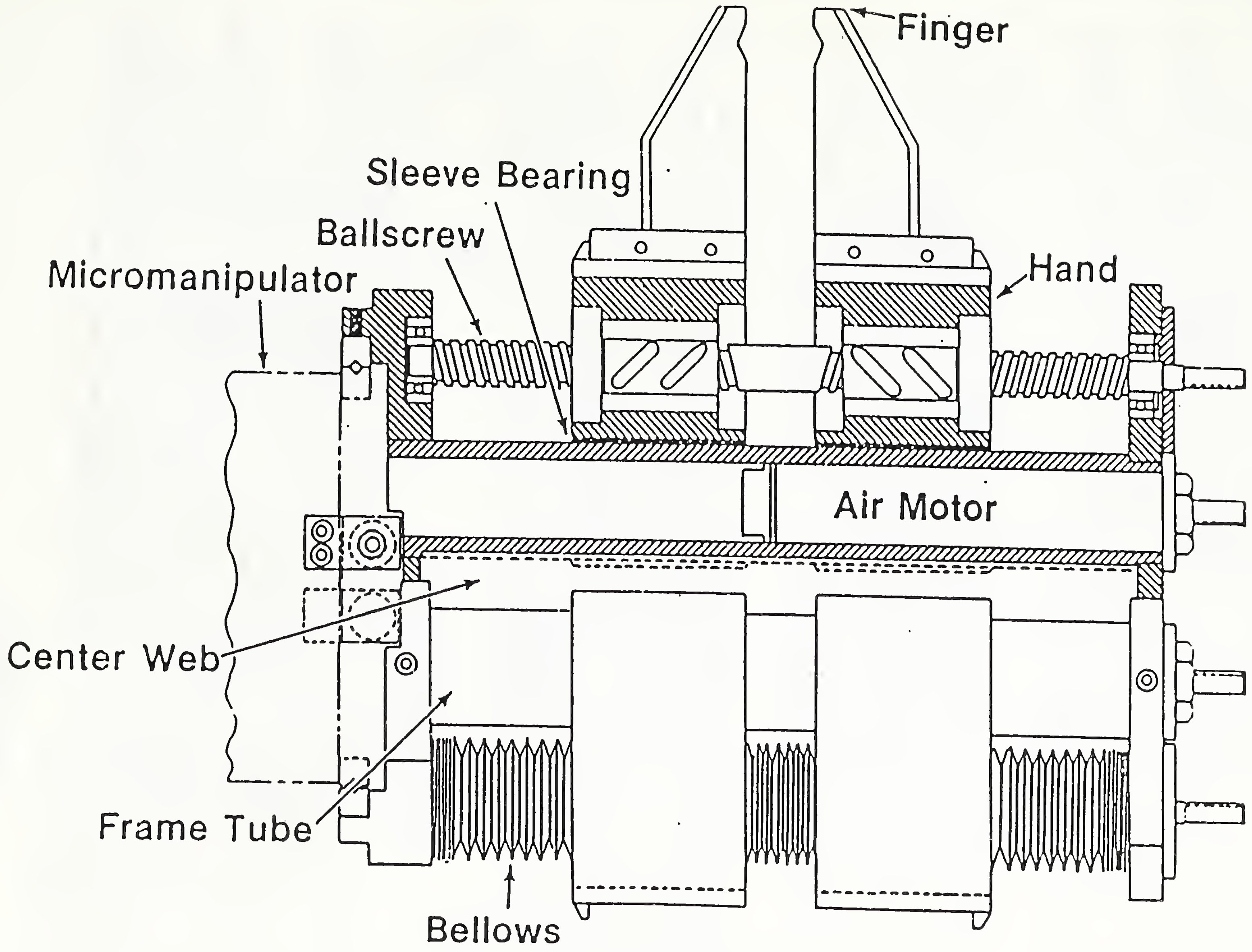


FIGURE VI.1. GRIPPER CONSTRUCTION

the center point of the gripping action changes due to the finger deflection by only a cosine error. Second, the dovetail arrangement allows the robot to easily exchange two fingers at one time with the help of a special fixture. The fixture is a rectangular plate with four aligned pegs extending from one face of the plate. Two pegs engage and hold one finger in place while the hand moves to detach itself from the finger. The basic finger changing operation consists of first positioning the gripper near the fixture with the jaw completely closed. Then the pins engage the fingers and hold them in position while the jaw opens leaving the fingers behind. The operation is then reversed to insert a new set of fingers.

The jaws are actuated via a ball screw and air-motor drive system. The ball screws are precision ground with a lead of 0.1 inch. Two separate helixes on either end of the ball screw are opposite in direction so that, as the ball screw turns, the jaws either open or close. One air motor is mounted inside each tube and is connected to the ball screw through a gear linkage. The air to the motor is controlled by a solenoid-powered direction control valve mounted separately from the gripper. A flexible bellows covering is used to protect each ball screw from contamination. This drive system is capable of exerting a closing force of 1000 lbf at an air pressure of 100 psi.

The position sensing system that determines the size of the jaw opening consists of three proximity sensors and an eight-pinned gear for each jaw. The gear mounted on the air motor has eight pins evenly spaced around the gear at a fixed radius. Two sensors are mounted so that as one pin crosses their path, one sensor sends a pulse back to the controller which is 180 degrees out of phase with the second pulse. The controller keeps track of the number of pulses being received and derives the jaw opening based on a "home" position. The home position is determined by opening the jaw until a third proximity sensor, mounted on the gripper end plate, is triggered. This configuration allows for a resolution of 0.0125 inch over the full 5.9 inches of travel range of each jaw.

1.4 Electronic Components

The heart of the electronic hardware is an 8-bit 8751 single-component microcontroller. The 8751 is a control-oriented computer that has an on-chip 128 bytes of data memory and 4K bytes of program memory. The control program is stored in UV-light erasable electrically programmable ROM. The microcontroller has four 8-bit parallel ports for I/O operations. Each bit in the ports is also individually addressable. In addition, the 8751 contains two 16-bit counters for measuring time intervals, measuring pulse widths, counting events and generating precise, periodic interrupt requests. It also has a serial I/O port configured in full-duplex mode to facilitate communications with a standard CRT terminal or other controllers. As well as communicating to other devices through the serial RS232C link, it can also communicate through single-bit parallel lines. A schematic diagram of the gripper controller is shown in Figure VI.2.

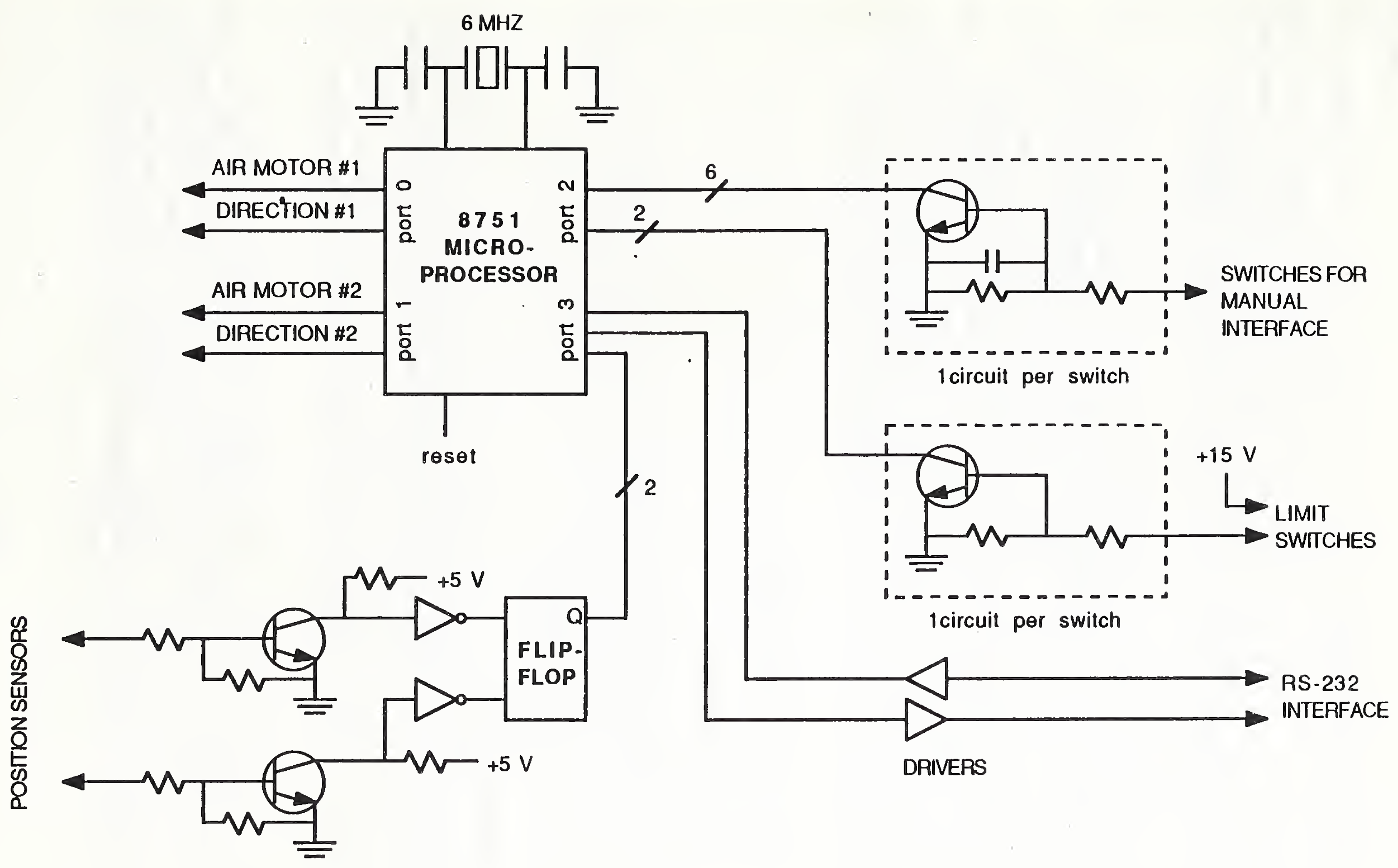


FIGURE VI.2. SCHEMATIC DIAGRAM OF THE GRIPPER CONTROLLER

1.5 Software Components

The software program is written in a high-level language, PLM, which allows for modularity, flexibility, and maintainability. The system program is composed of six main modules: initialization, command handler, interrupt handler, output and status handler, command driver, and the main control loop. Each main module contains a number of procedures to perform smaller functions.

The initialization module contains procedures to set up all the parameters for interrupt, serial port, parallel ports, defaulted force value, etc. The command handler decodes ASCII commands and interprets the command for proper procedure execution. The interrupt handler module sets different levels of interrupt to take care of the serial I/O, the position feedback pulse counters, and the control cycle. The output and status handler module consists of procedures that buffer characters for outputting, output ASCII characters, and output the status of the gripper. The command driver module activates the motor to stop and start, to position the gripper finger opening, and to reset the gripper. The main control loop module checks various limit switches, time out conditions, and position counter readings for controlling the overall operation of the gripper.

2. OPERATION OF THE DEVICE

2.1 Setup Requirements

setup requirements for the gripper controller are quite simple. ıunication is carried out through a 1200 baud serial link. On power up, the gripper automatically opens both jaws until the proximity sensor is triggered. At this location the encoder counters are set to zero and the gripper controller is ready to accept commands.

2.2 Operation Mode

The gripper can be operated in either remote or local mode, with remote mode as the default on power up. In remote mode, the gripper controller is connected to a workstation controller via an RS232C link. In the local mode, the operator communicates with the gripper controller via a CRT terminal. To view comments sent to the gripper controller on the CRT terminal, an echo mode command must be sent to the controller.

Both remote and local modes operate under similar conditions and sets of commands, and all of the responses are the same. Three basic commands are used to control jaw positioning. One command closes a jaw until the motor stalls, another opens a jaw until the limit switch is triggered, and the third is used to position the jaw at a prescribed opening. In addition, other commands are used to request status, perform a software reset, and echo commands back to the workstation controller or operator terminal.

The commands and their respective formats are:

```
O <n> CR
```

open gripper #n, where n is an ASCII character 1 or 2, and CR means carriage return;

```
C <n> CR
```

close gripper #n, where n is an ASCII character 1 or 2, and CR means carriage return;

```
P <n> XX CR
```

position gripper finger #n to an opening of XX millimeters, where XX are ASCII characters;

```
S <n> CR
```

request status of gripper #n;

```
E <1> CR
```

put gripper controller serial link in echo mode;

```
E <0> CR
```

put gripper controller serial link in nonecho mode;

```
R <n> CR
```

software reset gripper #n.

The responses to these commands include ready and busy signals when the controller is ready to receive a command or is busy executing one. An error condition response informs either the workstation controller or operator terminal which jaw the error has occurred on and what type of error has occurred.

The responses and their respective formats are:

```
R <n> CR
```

gripper #n is in the ready mode, and CR means carriage return;

```
B <n> CR
```

gripper #n is in the busy mode;

E <n> X CR

gripper #n is in the error mode, where X is the error code in ASCII characters as follows:

X=0, illegal command;
X=1, part dropped from gripper;
X=2, bad encoder counts;
X=3, gripper is opened wider than commanded;
X=4, tried to open, but won't move;
X=5, bad limit switch;
X=6, gripper is stuck;
X=7, tried to close, but won't move.

3. SUGGESTIONS FOR FUTURE DEVELOPMENT

Originally, force control of the gripper was thought possible by varying the air pressure to a pneumatic motor. However, due to the varying starting-torque characteristic of turbine-type pneumatic motors, predictably accurate force control could not be obtained. Pneumatic motors posed another problem: if not maintained properly by adding oil to the air supply line for lubrication, the motor eventually jams up. A good alternative would be high-torque DC brushless motors which require no maintenance. Although expensive, this type of motor can be easily designed into a closed-loop control configuration and thus provide accurate force control. With closed-loop sensing of the current through the DC motor, force control of the gripper fingers can be implemented without extra sensors like strain gages or pressure-sensitive sensors on the gripper fingers. Additionally, a position device is needed in a closed-loop control system to close the position loop. Proximity sensors are currently used for position feedback, and could be eliminated because the Hall effect sensors used in the DC brushless motors to supply commutation for the motor controller can also be used for encoder-type position feedback.

VII. PROGRAMMABLE STOP

The programmable stop is a microprocessor-controlled device used to provide a reference stop point for parts in the turning center and to push collets out of the spindle for removal by the robot. It is mounted behind the spindle in the turning center and uses a motor to drive a ballscrew into and out of the spindle.

1. PROGRAMMABLE STOP DESIGN

1.1 Overview

The implementation of a reference stop as a microprocessor-controlled, motor-driven leadscrew device allows for a compact, simple design which performs two functions. The first function is that of an adjustable, or programmable, reference stop for parts being machined in the turning center. Before the part blank is inserted into the collet, the programmable stop is sent out to a location specific to that part and operation. After the part blank is inserted by the robot , it is pushed to its final position against the end of the programmable stop. The collet is then closed, clamping the part in its correct position, and the programmable stop is backed away so it does not come into contact with the part (the part spins at high speed - the end of the programmable stop does not). The programmable stop is likely to be needed again to set the location of the part when the other side is machined.

The second function of the programmable stop is to push out loosened collets for pickup by the robot. Even after a collet has been loosened, there is enough friction to make it difficult for the robot to withdraw it smoothly. To alleviate this, the programmable stop is used to push the collet partly out of the spindle, past the point of binding, for the robot to remove. This also guarantees the robot a repeatable pick-up point for the collet.

Mechanically, the programmable stop consists of a ballscrew inside the spindle of the turning center, and is belt-driven by a servo motor. The end of the ballscrew closest to the face of the spindle is designed to be used as a reference stop for parts, and to push out collets and finished parts for robot pick-up. It has a linear travel of approximately seven inches. An incremental encoder linked to the ballscrew provides position information.

In addition to the microprocessor and support circuitry, the programmable stop's controller system consists of an incremental encoder for position feedback and a motor driven by a commercially obtained motor controller. The controller software is stored in EPROM and is written in PLM-51, a high-level language written for the 8051 family of Intel microprocessors. The software is composed of different routines used for position control, error checking, and communication with the workstation controller.

1.2 Control Architecture

The programmable stop is operated by using a basic servo-control algorithm. Upon receiving a position command from the workstation controller, the ballscrew is driven by the servo-motor in the appropriate direction. The incremental encoder is read and the counts are added or subtracted from the current position count until the correct position is reached. The motor is then stopped and the position is held.

The implementation of the programmable stop servo-control algorithm is accomplished by two feedback loops. An inner feedback loop consists of the motor, tachometer, and motor controller. An outer loop consists of the motor controller, an incremental position encoder, a D/A (digital-to-analog) converter, and a microprocessor.

The inner loop controls the speed and direction of the motor based on the command and tachometer signals to the motor controller. The motor controller (which was commercially obtained) outputs a high-voltage PWM (pulse-width modulated) signal to the motor. Based on the feedback from the tachometer (which is attached to the motor) and the command signal input, the motor controller varies the speed and direction of the motor.

The incremental encoder in the outer loop is directly linked to the motor, and is used to precisely count the number of motor revolutions, and thus the linear travel of the programmable stop. The encoder has a resolution of 2048 parts per revolution, and there is a 7:1 turns ratio between the motor/encoder and the ballscrew. Further, one revolution of the ballscrew translates into 0.1 inch of linear travel for the programmable stop. Thus, one revolution of the motor translates into approximately 0.014 inch of travel of the programmable stop, with a theoretical resolution of less than ten microinches.

The microprocessor keeps track of the number of counts from the encoder and compares this number to the destination number. When the destination is reached, the microprocessor sends a value to the D/A converter which corresponds to a voltage of zero. This zero voltage (at the output of the D/A) is the command signal, which is sent to the motor controller and stops the motor. To move the ballscrew forward or back, a non-zero voltage of the correct polarity (determined by the microprocessor) is sent from the D/A converter to the motor controller.

The number of encoder counts is relative to a "home" position, which is determined by a limit switch. Home position is determined automatically whenever the power to the microprocessor circuitry is turned on; it can be reset at any other time.

1.3 Mechanical Components

An illustration of the mechanical components is shown in Figure VII.1. The mechanical components of the programmable stop are a 0.1 inch pitch ball screw (1) which is driven by a DC servomotor (2) through a 7.2:1 timing pulley drive.

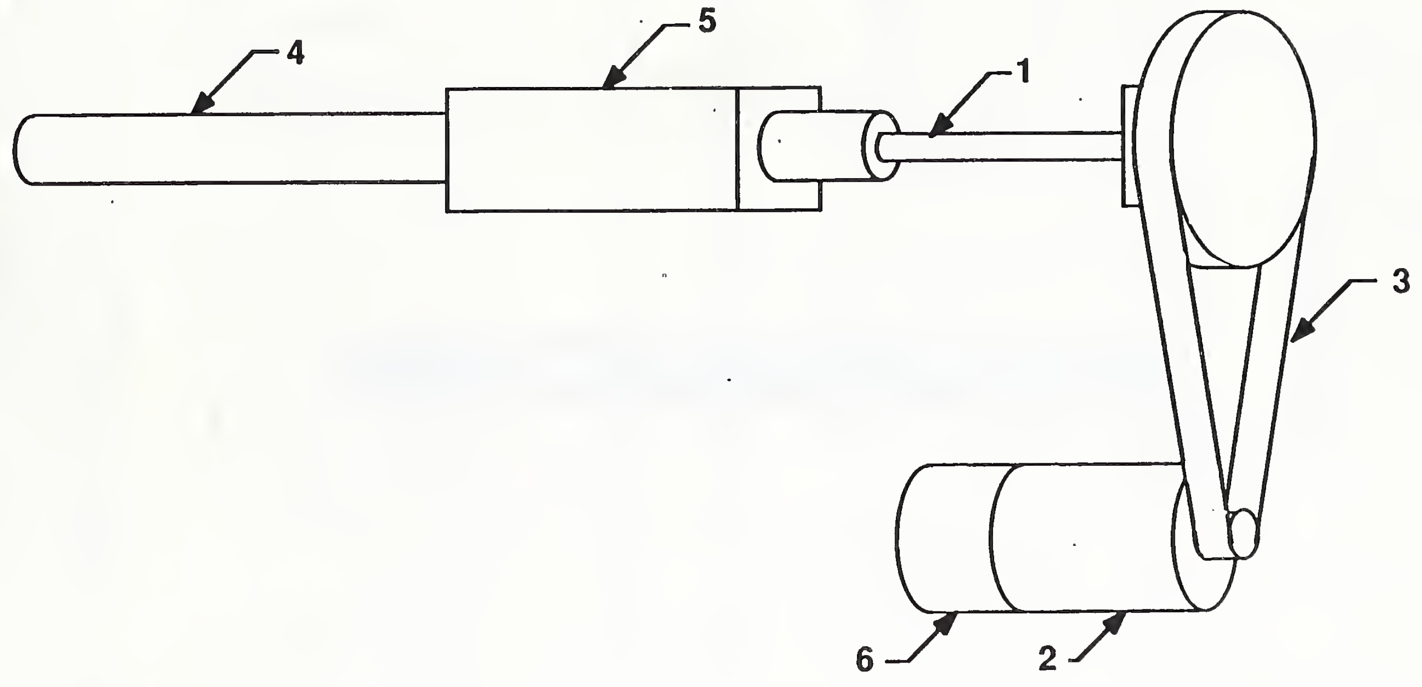


FIGURE VII.1. PROGRAMMABLE STOP - MECHANICAL COMPONENTS

The ball screw drive actuates a shaft (4) which is housed in a nonrotating, linear bearing (5). The shaft has a total travel of about 7 inches. The tip of the shaft is loaded with a stiff spring (not shown) to give it the compliance needed when the drawbar is closed. An 11-bit encoder (6) is coupled directly to the DC motor to avoid any backlash problems.

1.4 Electronic Components

The electronic components in the programmable stop controller (illustrated in Figure VII.2) consist of the microprocessor and its peripherals, an incremental-position encoder, the motor/tachometer unit, and the commercially obtained motor controller. There are also various switches and relays.

An Intel 8031 microprocessor is used for control. It has an 8-bit data bus, bidirectional ports, a serial communication port, timer/counters and external interrupt lines. There are 128 bytes of volatile memory on the 8031 for temporary storage, and program memory is stored in an EPROM (Erasable Programmable Read Only Memory). A 12-bit D/A converter, the AD567, converts the digital command from the microprocessor to a voltage command signal which is sent to the motor controller. A low offset operational amplifier (AD544) is used to buffer the output of the D/A converter.

The encoder used is an 11-bit absolute encoder. To simplify the design, and because of alias problems, the encoder is used as an incremental encoder by looking only at the least significant bit (LSB) line.

The motor employed is a permanent-magnet DC servo motor supplied with a tachometer. It will maintain a continuous torque of 35 ounce-in and a peak torque of 260 ounce-in.

The motor controller is a linear servo controller and is capable of supplying +/- 5 amperes of continuous output current to the motor. It accepts feedback from the motor tachometer and a control voltage from an external source; in this case the microprocessor control circuit. The motor controller also has an externally controlled motor shutdown line. This line is controlled by the microprocessor and is independent of the command voltage. The controller also has adjustments for gain, current-limiting, and balance. One micro-switch is used for setting home position, another is used as an end-of-travel limit switch (the motor is turned off when the switch is activated), and a relay is used by the microprocessor to activate the remote shutdown line on the motor controller.

Microprocessor control is implemented using the 8031 microprocessor with an 8192 (8K) byte EPROM for program storage. The incremental encoder information, the D/A input, and the assorted status and control lines are all multiplexed onto the 8-bit data bus of the 8031. The voltage output of the D/A converter controls the direction and speed of the ballscrew motor through the commercial motor controller.

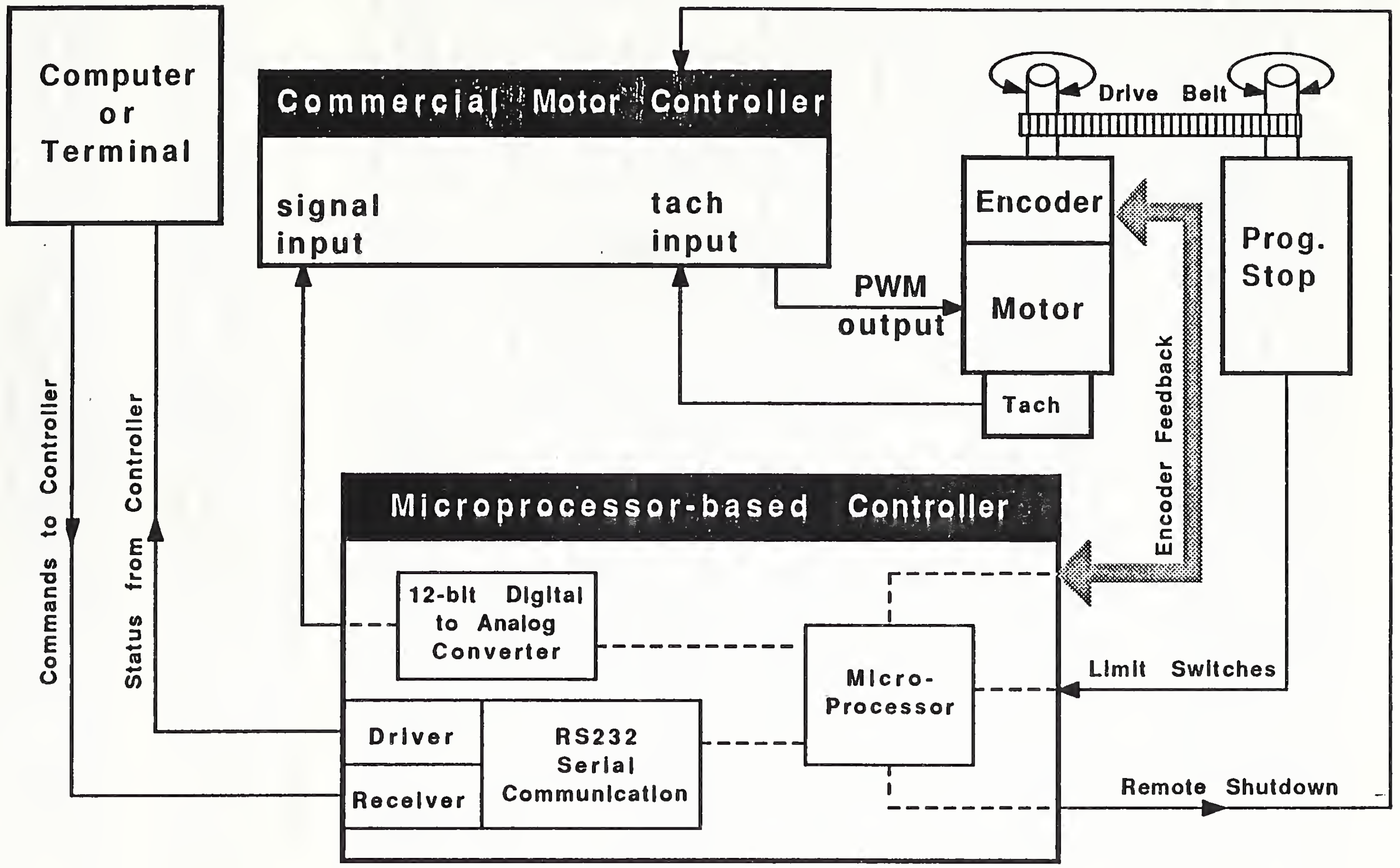


FIGURE VII.2. FUNCTION BLOCK DIAGRAM OF THE PROGRAMMABLE STOP CONTROLLER

When the power to the programmable stop controller is applied, the microprocessor sends the stop to its home position. This is done by causing the ballscrew to rotate in the direction which moves the stop away from the machining area. In the following discussions, "forward" will refer to the programmable stop moving toward the cutting tool in the machining area; "back", "reverse" or "toward home position" will refer to the stop moving away from the machining area.

The home position is determined by a limit switch. When the stop is fully retracted, the switch is contacted. During the travel toward home position, the encoder is ignored. Instead, the microprocessor uses one of its external interrupt lines to detect the closing of the home limit switch. As soon as this is detected, the microprocessor directs the motor to stop. After a brief pause, the motor is directed to turn the ballscrew a fixed number of revolutions to move the stop forward. This moves the stop off the limit switch and it is from this point that the encoder counts are referenced. This process can also be initiated manually by a reset button or by a software command.

The microprocessor communicates with the workstation controller over a serial (RS232) link. When the programmable stop controller receives a command from the workstation controller to move the stop to a certain position, the number of encoder counts necessary to move from the present position is calculated. The motor is moved in the proper direction (the present position may not be home) and the encoder is monitored. When the programmable stop is within one ballscrew revolution of the destination, the motor is slowed, then stopped at the destination. The remote shutdown is activated, disabling the motor. The remote shutdown circuitry is designed so that the normal state is for the motor to be disabled. The remote shutdown relay in its normal state (i.e., inactive) will disable the motor drive. In order to enable the motor drive, the remote shutdown relay must be activated. This was done as a precaution against an unforeseen runaway situation with the motor.

If, through error or other circumstance, the stop continues too far forward, then it will contact an overtravel limit switch. This switch is connected through a relay with the remote shutdown input of the motor controller, and the motor will stop.

1.5 Software Components

PLM-51, a high-level language for the Intel 8031 microprocessor, was used in writing the programmable stop controller software. The program is linked with the PLM-51 library, compiled and stored in EPROM, occupying approximately 3000 bytes of code.

PLM is a structured language, and this program is written mainly as a series of procedures. Each procedure defines a particular command, sequence of actions, or interrupt routine. The main program loop ties everything together by checking flags and calling the appropriate procedures.

The control software has a real-time control structure. The status of the programmable stop is checked continuously and a routine is used to implement serial communication. Within the program there are three interrupt procedures. One of the microprocessor's external interrupts is used to detect the closing of the "home" switch. Another interrupt is activated when the encoder pulse counter overflows. The third interrupt detects data flow into and out of the microprocessor's serial port register. When any of these interrupts are activated, the program control immediately jumps to the appropriate procedure for that interrupt. Among the necessary actions taken within the procedure is the setting of certain flags. When the interrupt procedure is completed, program control jumps back to the main program loop to continue where it left off. The flag or flags that were changed during the interrupt procedure are checked during the completion of that loop or the next one, and appropriate action is taken.

An example of an interrupt procedure is illustrated in the following. If a command was sent by the workstation controller to the programmable stop controller, a serial I/O (input/output) interrupt would be generated. Among other things, the serial interrupt procedure would set a "command pending" flag, which, when read during the main program loop, would initiate a routine to identify which command was sent, and how to respond to it.

The control program, in general, begins at power up with a setup procedure which disables the motor and initializes all variables and control registers. Next, flags are set, which, when checked in the main loop, will direct the stop to be brought back to the home limit switch in order to set the home position. The home set routine occurs only once at each power up, and the remainder of the program is a repetitive loop.

Flags are checked to determine if a character in the output buffer is waiting to be sent (a response to the workstation controller, for example) or if there is a command pending. The proper motor speed and direction are determined, and the encoder counter status is checked to see if the stop is getting close to its destination. Finally, the D/A converter is updated, and the proper polarity and magnitude control signal is sent to the motor controller.

2. OPERATION OF THE PROGRAMMABLE STOP

2.1 <u>Local Operation</u>

2.1.1 Setup Requirements

An RS232 terminal is used for local control and operation of the programmable stop. Incoming data is received on line 2, and data is sent to the terminal over line 3. Line 7 is used as the signal ground, as is standard. The terminal should be set to full duplex mode, no parity checking, and with a baud rate of 1200. In local mode, the characters are echoed back to the terminal, and a carriage return/line feed response is included.

2.1.2 Commands and Responses

The programmable stop commands are listed below. The ENTER (or carriage return) key must be appended to complete each command, because the software looks for it before setting the "command pending" flag in the program loop. If the command includes numeric parameters, they are indicated within < >. The standard response to any command from the workstation is a "b" (for "busy") upon receipt of the command, and an "r" (for "ready") after completion of the command. In local mode, these character responses are each followed by a line feed (LF) and a carriage return (CR).

In the following discussion, the quotation marks shown are not included with the command, nor are the blank spaces. For example, "b LF CR" is actually sent bLFCR , and gh CR is entered ghCR.

ec <0 or 1> CR   Echo

This command sets or resets the ECHO flag. When the ECHO flag is set (1), which it would be for local operation, the command sent to the programmable stop is echoed back to the terminal screen, with a LF (line feed) character. When reset (0), there is no echo or LF. The response to this command is a "b CR LF" upon receipt and a "r CR LF" after completion.

gh CR   Go Home

This command sets a flag which causes the stop to move back to the home limit switch and initialize home position. The response to this command is a "b CR LF" upon receipt and a "r CR LF" after completion.

gl <0 to 535> CR   Go to Location < >

This command moves the stop to the absolute position specified by the number. The positions are referenced to home position, which is 0. The stop can travel either forward or back to reach the position. A unit length of travel is approximately 0.014 inch. The response to this command is a "b CR LF" upon receipt and a "r CR LF" after completion.

sd <0 or 1> CR   Shutdown the Motor

If a 1 is entered, this command sets a flag which deactivates the output stage of the motor driver, disabling the motor. If a zero is entered, the motor is enabled. The response to this command is a "b CR LF" upon receipt and a "r CR LF" after completion.

As for the programmable stop responses, there are four error messages that can be sent back to the workstation controller. The messages have the format "e <number> LF CR" , where <number> is the error type. Again, the quotation marks and spaces are not included.

One error condition is an unrecognizable command. This usually occurs when the command is misspelled. Another is when an illegal numeric parameter is used, for example, a number greater than 535 with the "gl" (Go to Location) command. A third error condition is if the home limit switch is activated without the command to do so having been given. The fourth condition is if the overtravel limit switch is ever activated.

2.2 Remote Operation

2.2.1 Setup Requirements

Remote operation entails communication with, and control by, the workstation controller. The setup requirements are similar to those for local operation, and the same RS232 connector is used. Data to the workstation controller is transmitted on line 3, data is received on line 2 and line 7 is signal ground. The difference is that there is no echo back to the workstation controller and the LF character is not included in the responses.

2.2.2 Commands and Responses

The commands are the same in remote mode as in local mode. Again, the CR character must follow the command, and the spaces and quotation marks are not included. The responses, "b" and "r", are the same also, except the LF character is not included.

ec <0 or 1> CR   Echo

This command sets or resets the ECHO flag. When the ECHO flag is set (1), which it would be for local operation, the command sent to the programmable stop is echoed back to the terminal screen, with a LF (line feed) character. When reset (0), there is no echo or LF. This is the normal power up state, and the workstation controller does not normally need to use this command. The response to this command is a "b CR" upon receipt and a "r CR" after completion.

gh CR   Go Home

This command sets a flag which causes the stop to move back to the home limit switch and initialize home position. The response to this command is a "b CR" upon receipt and a "r CR" after completion.

gl <0 to 535> CR   Go to Location < >

This command moves the stop to the absolute position specified by the number. The positions are referenced to home position, which is 0. The stop can travel either forward or back to reach the position. A unit length of travel is approximately 0.014 inch. The response to this command is a "b CR" upon receipt and a "r CR" after completion.

sd <0 or 1> CR   Shutdown the Motor

If a 1 is entered, this command sets a flag which de-activates the output stage of the motor driver, disabling the motor. If a zero is entered, the motor is enabled. The response to this command is a "b CR" upon receipt and a "r CR" after completion.

st CR   Start

This command is used as a software reset. It reinitializes all the variables and registers, and sends the stop back to reset the home position. The response to this command is a "r CR".

The same error messages are used in remote mode as are used in local mode. The messages have the format "e <number> CR" , where <number> is the error type. Again, the quotation marks and spaces are not included. See the error description in the local operation section, above, for a list of errors.

3. SUGGESTIONS FOR FUTURE DEVELOPMENT

As mentioned previously, the theoretical resolution of the programmable stop is quite good. The encoder resolution is 2048 pulses per revolution, and there is a 7.2:1 ratio between encoder and ballscrew revolutions (the encoder is directly coupled to the shaft of the motor). Since one revolution of the ballscrew translates into 0.1 inch linear travel for the stop, the final resolution (ideally) is about 7 microinches. One of the largest limiting factors is the repeatability of the home limit switch - probably the weakest link in the system. Since the home (reference) position is directly linked to this switch, the stop position repeatability from one home reset to the next is limited by the switch. The actual repeatability is closer to 0.001 inch. A better switch or other sensing device would improve the overall repeatability of the stop.

VIII. TOOL SETTING STATION

For automatic tool setting and tool-wear monitoring of cutting tools on the CNC turning center, a Linear Variable Differential Transformer (LVDT)-based sensor system is used. This sensor system provides information on the location of the cutting edge of a tool with respect to the machine reference. In order to do this, the cutting edge of a tool depresses one of three plungers. The LVDT provides feedback corresponding to the displacement of each plunger. The feedback information is then used to determine the location of the cutting edge. This operation is performed several times throughout the working life of a tool, the first of which is the original adjustment when the tool is initially placed in the machine. Checks are made periodically to compensate for tool wear and to detect when excessive wear has been reached.

1. DESIGN OF THE TOOL SETTING STATION

1.1 Overview

The mechanical system of the tool setting station consists of an LVDT-based gage head and a hydraulically actuated, precision four-bar mechanism for positioning the gage head. The system was designed to satisfy many requirements, but the main goal was to achieve a repeatability of at least 50 microinches. This was accomplished by proposing a design and performing an error budget analysis on the design to predict the repeatability of the system and to modify design parameters if necessary. Results from tests on the system show that the repeatability requirement was achieved.

The controller hardware consists of two printed-circuit boards designed in house. One board provides reference signals needed by the LVDT, signal conditioning, and analog-to-digital (A/D) conversion. The other board is an Intel 8088-based computer with an Intel 8087 math coprocesser. Interfaced to the computer are the valves and sensors needed to control the four-bar mechanism, cover, and air.

All of the software for the controller consists of one module written in PL/M and some general I/O routines which are linked to the module with no operating system. Because the device performs only one main function, no operating system was needed. Separate procedures are written for each operation and error checks are included where appropriate. The latching and inputting of the A/D data is initiated by an interrupt produced by the A/D chip. A routine is included to convert the digital data from the A/D converter to geometric units using the calibration coefficients.

1.2 Control Architecture

The basic control algorithm consists of performing an operation upon receiving a command from the machine controller. Once the command is completed, the tool setting station controller sends a "ready to receive" command back to the machine controller. Examples of the types of operations are the actuation of the four-bar mechanism and sending out the linearized LVDT data.

1.3 Mechanical Components

The design of the tool setting system is based on two mechanical subsystems. The first is a gage head for providing the actual measurement, and the second is a mechanism to position the gage head in the working volume of the machine. The gage head is a unique precision design using a state-of-the-art displacement transducer as the reference. A Linear Variable Differential Transformer (LVDT) was chosen as the sensor. An LVDT has several attributes which make it ideal for a tool setting application. Its characteristics include high resolution, large gaging range, and microinch repeatability. In addition, the rugged construction of an LVDT permits it to function even after exposure to the substantial shock loads and high vibration levels often encountered during machine tool operation. The particular LVDT used has a repeatability of 4 microinches, a gaging range of +/- 0.06 inches, and can operate in temperatures ranging from -40 to 212 degrees F. The shaft of the LVDT is coupled to a non-rotating probe shaft. The probe shaft rides in a linear ball bearing assembly to minimize friction and eliminate the effects of radial play. The LVDT also incorporates a helical spring for extending the probe shaft and a dust cover to prevent foreign materials from entering the bearing assembly.

One disadvantage of using an LVDT in this type of application is that it only supplies a one degree-of-freedom measurement. However, it is necessary to gage tools approaching from any one of three directions: +X, -X, and -Z direction. If a separate LVDT were used for each direction, the resulting gage head would be expensive and rather bulky. Instead, a compact gage head was designed to provide displacement measurement in all three directions with a single LVDT.

The gage head (as shown in Figure VIII.1) consists of a square cam mounted on a shaft which is free to rotate about the y-axis. The probe shaft of the LVDT acts as a cam follower so that any rotation of the cam is detected by the LVDT. The cam is driven by depressing any one of three, cylindrical, nonrotating plungers (see Figure VIII.2), which are oriented so that tools approaching from any of the three measuring directions can be set. Each plunger rides in a bushing which is honed to tight tolerances to allow for a precision sliding fit between bushing and plunger. The shaft is mounted on ultra-precision, ABEC-9, preloaded ball bearings. Such high-precision bearings are used to minimize errors that result from radial and axial play.

The position of the gage head with respect to the machine tool reference will vary due to thermal gradients along the machine. The magnitude of this variation can be significant depending on the placement of the gage head. For example, tests conducted on the turning center at the AMRF show that continuous operation of the spindle and slides over a 500 minute period produced a temperature rise of 8 to 12 degrees C. (15-20 degrees F.). This temperature rise caused errors of the order of 2 micrometer/deg C. (40 microinch/deg F.) in

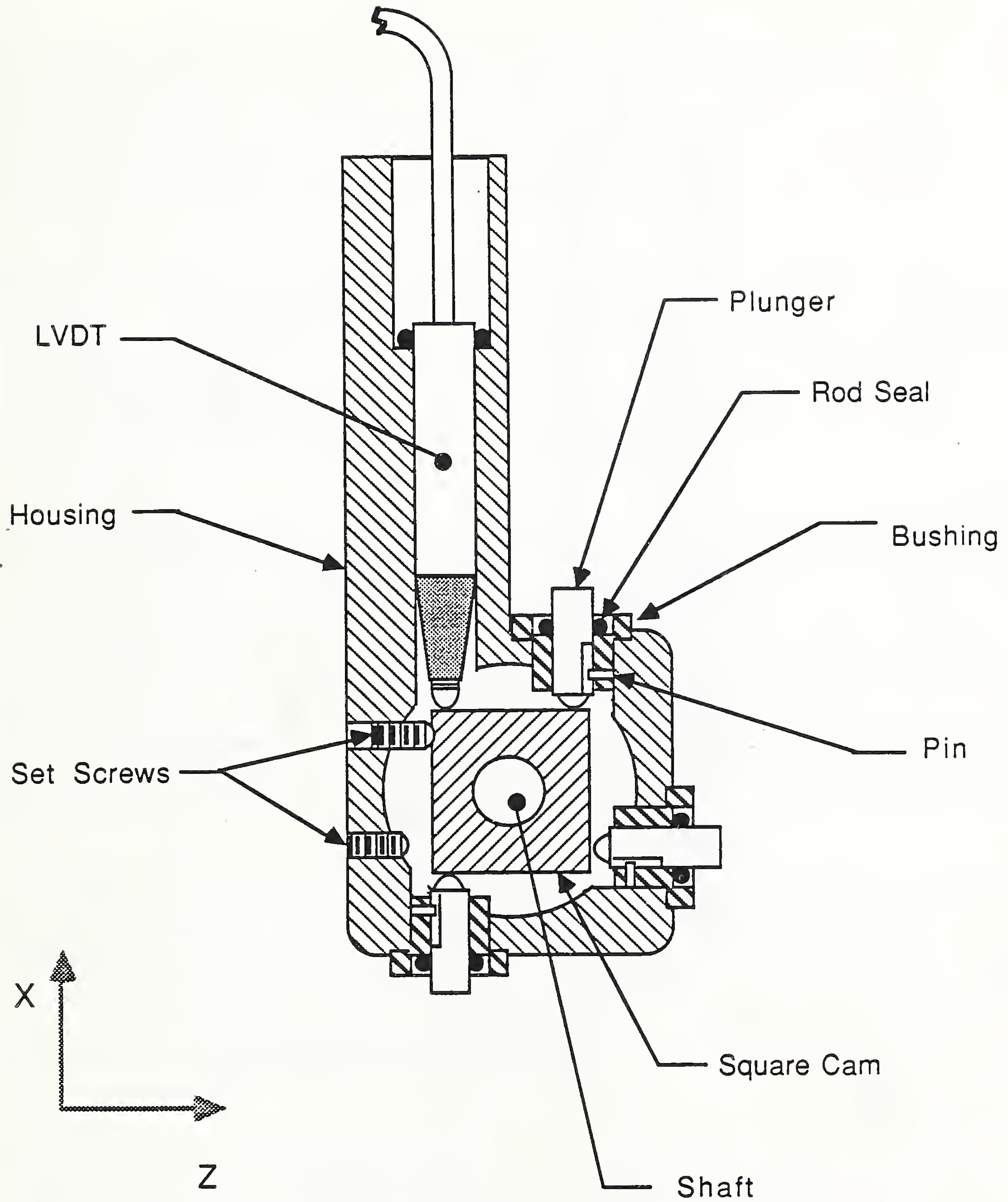


FIGURE VIII.1. CROSS SECTION OF GAGE HEAD

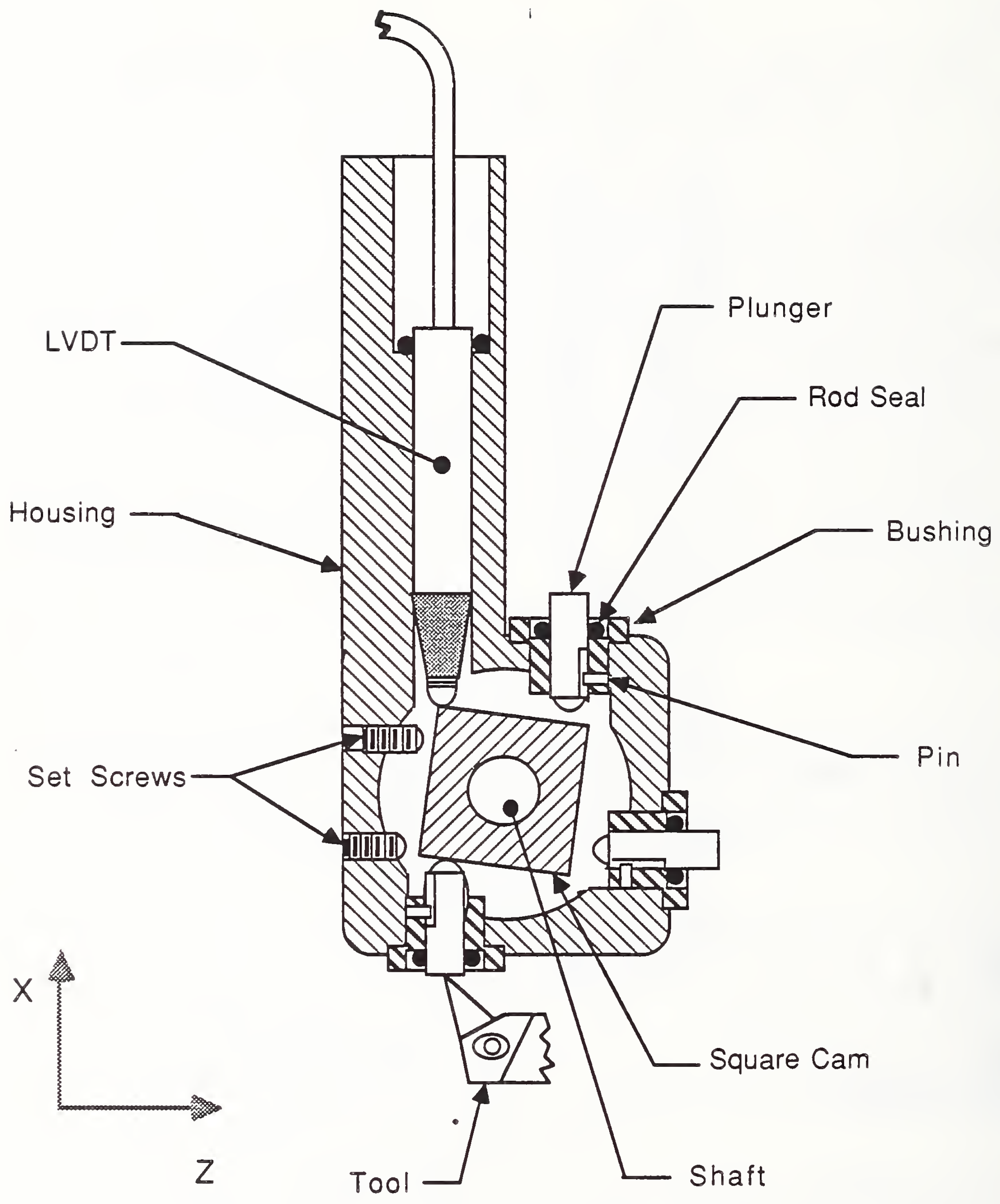


FIGURE VIII.2 OPERATING MODE OF GAGE HEAD

the radial direction and 20 micrometers/deg C. (400 microinch/deg F.) in the axial direction. One possible way of compensating for these errors is to measure the distance between the gage head and the machine tool reference surface before each setting operation. This measurement could be made using a second gage head mounted in a tool position on the turret. This method, however, adds expense, increases the setting cycle time, and reduces the flexibility of the machine tool.

The tool setting system design attempts to minimize the thermally induced errors by reducing the thermal loop between the gage head and the machine tool reference. To accomplish this, a four-bar mechanism, mounted on the spindle cartridge, is used to accurately position the gage head near the reference axes and within the working volume of the machine. In this way, the four-bar mechanism and gage head should closely follow the thermal growth of the spindle, both radially and axially. A second function of this mechanism is to ensure that the gage head does not interfere with the normal operations of the machine while not in use.

The four-bar mechanism, as shown in Figure VIII.3, is driven by a hydraulic actuator mounted on the head stock of the machine. The follower of the four bar is an arm which is attached to the gage head at one end. The other end of the arm is mounted on a shaft between two ultra-precision, ABEC-9, preloaded, ball bearings. The bearing assembly is housed in a base structure which is fixed to the spindle cartridge. In this configuration, the gage head can be rotated into an idle position when not in use, and into a gaging position for performing tool setting, as shown in Figures VIII.4a and VIII.4b, respectively. The arm is sufficiently long so that, in the gaging position, there is a considerable gap between the gage head and the spindle face. This gap allows many types of spindle tooling, such as collets and some step chucks, to remain in the spindle while tool setting.

To ensure the repeatability of the gaging position, many aspects of the four-bar mechanism design must be considered. For instance, the errors associated with the rotating elements are minimized by using ultra-precision ball bearings. Thermally induced errors are reduced by mounting the system as close to the machine reference as possible. If necessary, the thermal growth of the system can be compensated for by monitoring the temperature at a representative point on the device and predicting the amount of movement of the gaging position with respect to the machine reference. The prediction can be based on either experimental data or a finite element analysis of the system.

One other major design criterion affecting the accuracy of the system deals with the method of stopping and locating the arm in the gaging position. Rather than rely on the repeatability of the stop in the hydraulic cylinder, the four-bar is stopped by metal-to-metal contact between the arm and spindle cartridge. The first concern with this stop is placement. If any runout in the bearings exists, depending on the placement of the stop, these errors could be amplified at the end of the gage head. By placing the stop as far from the rotational axis of the arm as possible the runout errors are minimized. A second design consideration is to ensure that the stop does not overconstrain

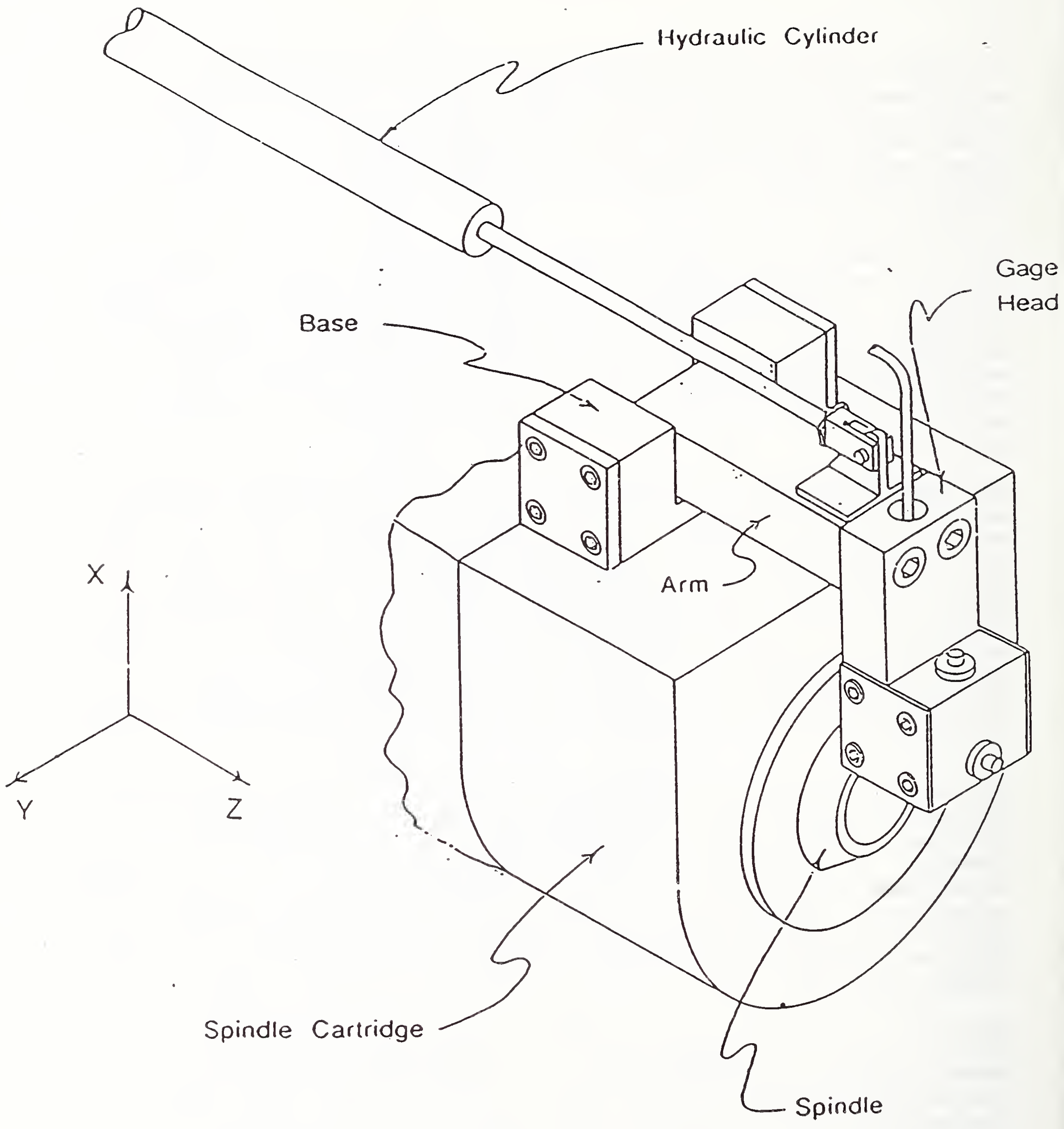


FIGURE VIII.3 ISOMETRIC VIEW OF TOOL SETTING SYSTEM

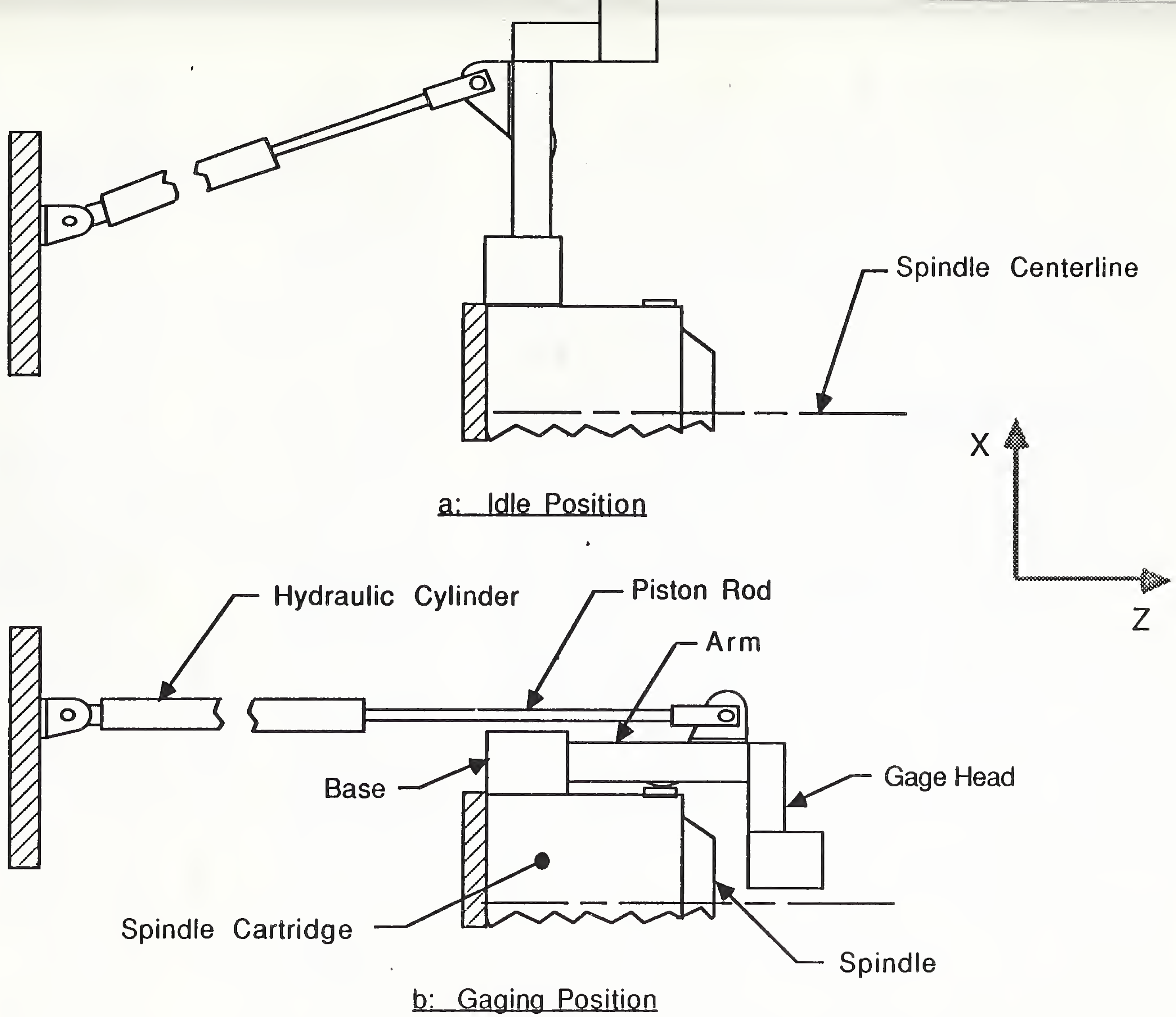


FIGURE VIII.4 POSITIONING MECHANISM IN IDLE AND GAGING POSITIONS

the system. The arm is already constrained in five kinematic degrees of freedom, and the only remaining constraint is the rotation about the y-axis. For this reason a sphere and flat plate configuration is used. An added advantage of this configuration is due to the resulting small contact area between the stops. Since the stops are subjected to coolant and oil, it is necessary to clean them periodically. The smaller the contact area, the easier it is to clean the surface effectively. However, if too small a contact area is used, the contact stresses that develop may exceed the elastic limit of the material. By choosing the appropriate spherical radius, material and material hardness, acceptable levels of contact area and contact stress can be obtained.

The rate at which the arm is lowered into the gaging position is also an important consideration for two reasons. The first is the repeatability errors that would result if the arm were to rotate too fast and come to an abrupt stop. These errors may stem from a plastic deformation of the stops or from vibrations due to the impact loading conditions. The second reason is the amount of time it takes for the arm to rotate from the idle to gaging positions. If the arm were to rotate slowly so that there would be no impact loading problem, then the setting operation would require too much time. The solution is to use a hydraulic cylinder equipped with a cushion at either end of the cylinder. The cushion acts to gradually decelerate the piston during its final one inch of stroke, and does not effect the piston's speed at any other time. In addition, a flow control valve is used to adjust the flow of fluid to the cylinder. Thus, the combination of cushioned cylinder and properly adjusted flow will minimize the stop impact without a drastic increase in gaging cycle time. Also used to control the direction of fluid flow is a direction control valve. All of the hydraulic components operate at a pressure of 500 psi.

Maintaining a clean stop surface is critical for ensuring the repeatability of the system. To clean these surfaces a blast of high-pressure, thoroughly filtered, and dry air is directed across the contact area. The four-bar mechanism is completely enclosed and protected from contamination while in the idle position. Before the gage head can be lowered into the gaging position a door to the enclosure is opened with an air cylinder. Two proximity sensors are utilized to determine whether or not the arm is in the idle or gaging position.

1.4 Electrical Components

The tool setting station controller consists mainly of an analog circuit board and a microprocessor-based computer board. The primary function of the analog circuit board is to convert the signal from a displacement transducer, into a 12-bit binary number. A block diagram of the circuit is in Figure VIII.5. In order to simplify the circuit description, it will be divided into three sections. The subjects, in the order they are described, are reference circuits, signal conditioning, and analog-to-digital conversion.

In this circuit there are two reference signals needed, one DC and one AC. To get the two reference signals, a 3 MHz crystal-controlled oscillator sends a

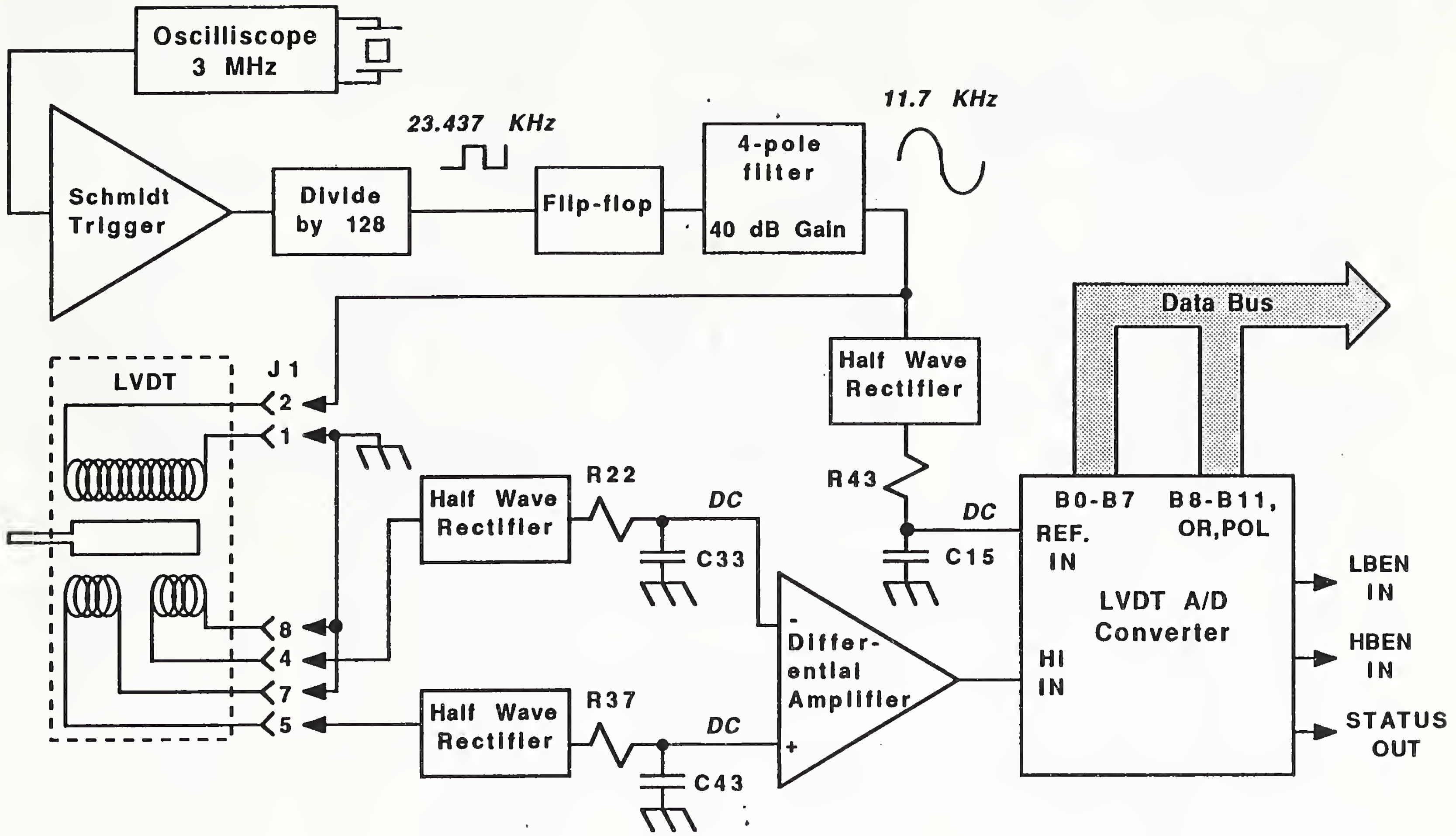


FIGURE VIII.5. FUNCTIONAL BLOCK DIAGRAM FOR THE TOOL SETTING STATION

signal to a Schmidt trigger circuit. The signal out of the Schmidt trigger is now shaped more like a square wave. This signal is divided by 128 and sent to a flip-flop. The flip-flop ensures that the signal will be symmetrical and also divides the frequency by two. Coming from the flip-flop is a 11.7 kHz square wave. This square wave from the flip-flop is filtered and amplified, resulting in a sinewave output that is a constant 11.7 KHz. The 11.7 kHz sinewave is used as the reference signal on the primary windings of the Linear Variable Differential Transformer (LVDT) displacement transducer. It is also half-wave rectified and converted to a DC signal for use as the reference for the LVDT A/D converter.

There are two sinewaves coming in from the LVDT. When the LVDT is in its zero position, both signals are identical. Each of these inputs are half-wave rectified and converted into a DC signal. The two resulting DC signals are sent to a differential amplifier. Any movement of the LVDT from its zero position will be seen by the LVDT A/D converter as a + or - DC voltage depending on the direction of movement.

The A/D converter runs continuously. At the end of each conversion cycle it sends out a signal called "status". The status line tells the microcomputer when the data is valid. The A/D converter is made to interface with an 8-bit bus with the converter's output tri-state buffered so the converter and bus can be tied together. It is a two step operation to read the data from the converter. When a low signal is sent into Low Byte Enable (LBEN) the output of the converter equals the low 8 bits of data. A low signal into High Byte Enable (HBEN) causes the output to be the next 4 higher bits of data, plus Over Range (OR) and Polarity (POL). The output enable commands can only be used one at a time.

The computer board has a large number of capabilities, the foremost of these being fast floating-point arithmetic that can handle 80-bit numbers. There are 34K bytes of memory available on this board, of which some can be nonvolatile memory. To communicate to external devices there are two input ports, two output ports, and a serial port. To help in real-time operation, there are six external interrupts. Figure VIII.6 is a block diagram of the microcomputer circuit board. As an aid in the understanding of the overall concept of the microcomputer circuit board, the following discussion is divided into four sections, each one containing a number of circuit descriptions. The subjects, in the order described, are microprocessor, control, memory, and Input/Output (I/O).

Two microprocessors are employed on this board. An 8088 in the maximum mode is used as the primary Central Processing Unit (CPU), and an 8087 Numeric Data Processor (NDP) is used as a coprocessor. In the maximum mode, the 8088 can address 1 megabyte of memory. When an 8087 is used in this manner, the programmer generally does not see it as a separate device; rather, the computational capability of the 8088 appears greatly expanded. The 8087 adds a fast floating-point arithmetic capability to the circuit. An increase in speed of 50 to 100, depending on the type of function, can be expected. Data as

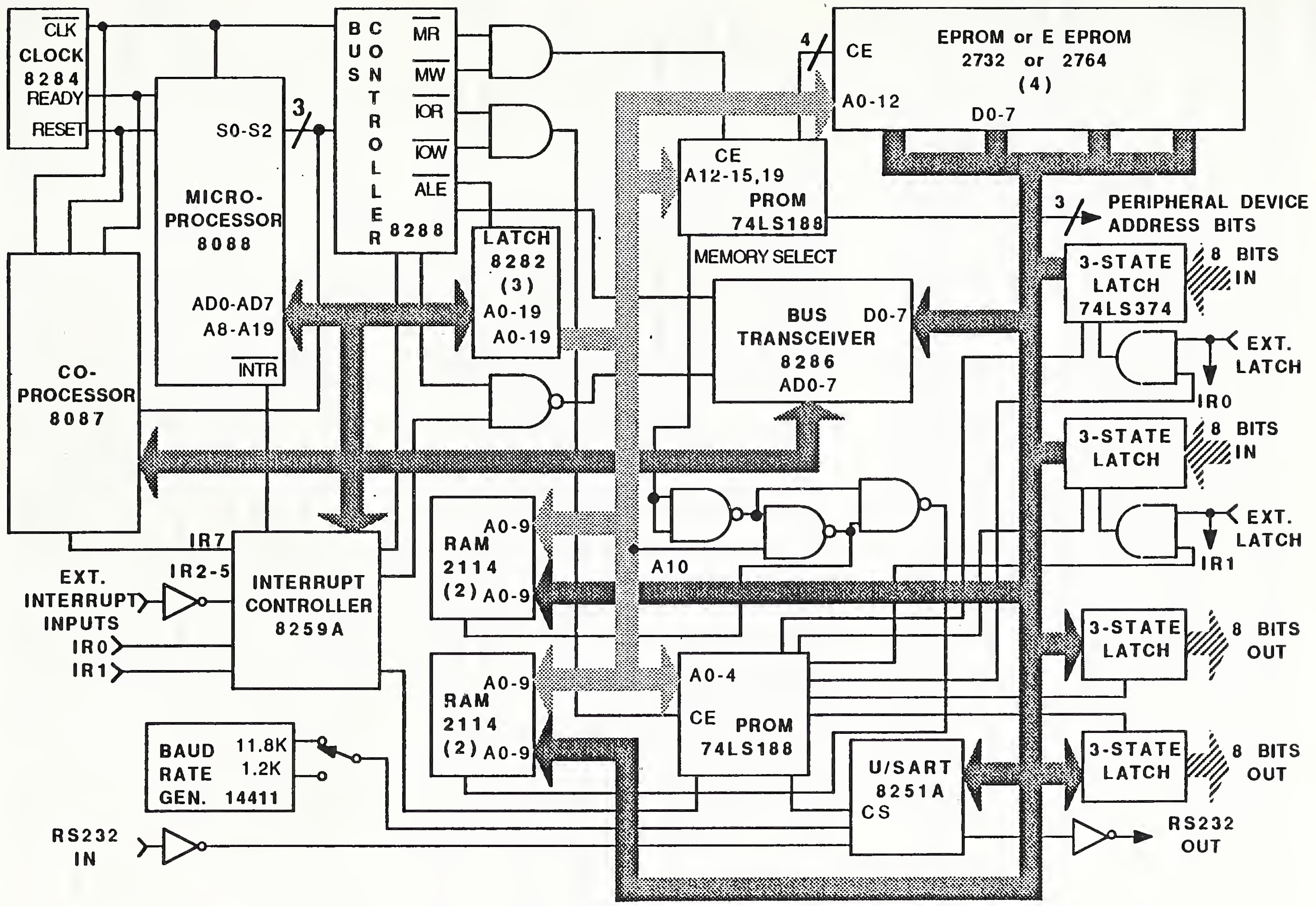


FIGURE VIII.6. SCHEMATIC DIAGRAM FOR THE NBS-DESIGNED MICROCOMPUTER

large as 80 bits, providing 19 digit mantissas and exponents up to 999, are valid values for the 8087 NDP.

Several integrated circuits comprise the control circuit. Two crystal-controlled circuits are used to generate the clocks needed by the microprocessors and the serial I/O circuit. The status of the microprocessor is monitored by a bus controller. Using this information, it generates the needed memory and I/O control command signals. Three latches use the address latch enable signal (ALE), supplied by the bus controller chip, to create the address bus. Programmable read-only memory (PROM) chips and a NAND gate decode the address bus then enable the proper memory or I/O chips. Because the CPU uses a multiplexed bus, a transceiver is needed to isolate the address/data bus from the data bus. An interrupt controller circuit is used to establish interrupt priorities when both an 8087 and interrupts are to be used by the CPU. In this design, there are six interrupts available; two of these are also external input commands that latch the input data bus.

There are three different types of memories used, random-access memory (RAM), erasable programmable read-only memory (EPROM), and electronically erasable programmable read-only memory (EEPROM). 2K bytes of RAM memory is available for use. The amount of EPROM memory available is determined by the type used. If 2732's are used, without using an EEPROM, there will be 16K bytes of EPROM, when using 2764's there are 32K bytes of EPROM. The advantage of EEPROM memory is that it can be addressed in a similar manner as RAM, but unlike RAM memory it will retain its data when power to the chip is removed. The disadvantages are slow writing speeds and low chip byte densities. A 2816 EEPROM was used because of its almost pin-for-pin compatibility with the 2732's and 2764's. If a 2816 EEPROM is used, the total amount of EPROM is reduced by 4K when using 2732's, and 8K when using 2764's. The reason for the EPROM size change is that the EEPROM uses one of the sockets normally used by an EPROM chip. The 2816 EEPROM will hold 1K bytes of data. In the latest version of the device, battery backed-up RAM was substituted for the EEPROM. This replaced the EEPROM memory on a chip-for-chip basis and offered the advantage that no special procedures were required to write to this RAM, which, in operation, is indistinguishable from the other RAM.

There are two types of I/Os used on the CPU board: serial and parallel. The serial input/output is accomplished by using a programmable Universal Synchronous/Asynchronous Receiver/Transmitter (USART). This circuit converts 8-bit parallel data to 8-bit serial data for transmission. It will also convert received serial data to parallel data. There are two baud (bytes per second) rates available: 1.2K or 4.8K. Interfacing to a terminal, using buffers to change the logic levels, is the function of this serial I/O circuit. Four 8-bit parallel ports make up the parallel I/O circuit, two input and two output. The two input ports data can be latched internally or externally, then read by the CPU at its convenience. The two output ports are bit-addressable by the CPU.

Interfacing to the sensors and the control circuits is done through the parallel I/O ports of the computer board. The output control lines go through current amplifiers to relays, which are used to switch servo-valve control voltages on and off. The four-bar mechanism, cover and air blast functions are controlled this way. Inductive position sensors are used to sense when the four-bar mechanism is up or down, and when the cover is open or closed.

1.5 Software Components

The controller software consists of one main program with two modes of operation, local and remote. Remote mode is obtained by sending a "#" on power up after which the device waits to respond to commands which consist of one character sent over the serial port. Local mode is obtained by sending the device a control-A on power up after which the device will proceed through a series of questions or statements requiring responses through the serial port. Local mode provides the means by which the linearization coefficients and range limits may be entered or modified. A list of all of the procedures in the operating program along with a short description of the function of each procedure follows:

AUTO:	If a character is waiting at the serial port, input the character. If the character is an illegal command, set a flag and send a binary 0 to the serial port.
BASE:	Send to the serial port the value of the A/D input in either binary or hexadecimal base. If a flag is set, the base is binary, otherwise it is hexadecimal.
COEF:	Display the twelve coefficients, for each of three directions, used to linearize the LVDT and allow for the changing of any of the coefficients. Display and allow for the changing of the range limits which determine which set of four coefficients are used in any specific linearization.
DROP:	Lower the contact arm if it is not presently making contact.
HELP:	Provides a list of the acceptable commands.
IDLE:	Check serial port for input. If there is no input, then return. If there is a legal command input, then service the command.
INTERP:	Convert a register reading into an appropriate integer.
INTER_U:	An interrupt procedure that inputs the LVDT reading.
LOC:	Set the reference to the coefficients for the current direction.
NEW:	Wait a preset length of time while checking the serial port for an input. Then take a new LVDT reading and process the reading.

RAISE:	Raise the contact arm if it is not presently making contact.
READ:	Enable a new LVDT reading to be taken. Wait for the reading to be taken. Disable further readings and convert the present reading to an appropriate integer.
READ_ARM:	Determine whether the arm is up or down.
TRANS_LVDT:	Convert the integer obtained from the reading of the LVDT and its interpretation into a floating-point number representing a distance based on the appropriate linearization coefficients.
YES_OR_NO:	Repeatedly send a message to the serial port until a response of "Y" or "N" is received. If the response is a legal command, the command is processed prior to the repetition of the initial message.

2. OPERATION OF THE TOOL SETTING STATION

2.1 <u>Setup Requirements</u>

Setup requirements are basically the same for both remote and local mode. The one major difference is that all the calibration information must be entered in local mode and then stored to make it available for use in remote mode. Communication in each mode is through a 25-pin serial line at either 1200 or 2400 baud.

2.2 <u>Commands and Responses</u>

2.2.1 Local Mode

In local mode, communication between the dumb terminal and the microprocessor is carried on in English, primarily with the microprocessor asking questions in English and the operator at the terminal responding with "Y" or "N". In addition, the microprocessor can occasionally request the input of parameter values, in which case an ASCII representation of the number is accepted by the microprocessor. In this mode, everything that the operator types at the terminal is echoed to the screen in order to facilitate human communication.

Local mode is obtained by sending a control-A to the serial port of the device after power up. The following pseudocode displays the operation under local mode. All messages sent by the detector are enclosed in single quotation marks and station responses in double quotation marks with items within square brackets [ ] being optional. Items between which choices must be made are separated by the vertical bar | while dummy values are shown as < > and non operable comments will be shown within braces { }. Numbers followed by a colon , ":", are labels.

```
0:      'LOAD A CALIBRATION?'
        "Y" | "N"
        if "Y"
                'LOAD THE PREVIOUS CALIBRATION?'
                "Y" | "N"
                if "N"
                      'LOAD THE BACK UP CALIBRATION?'
                      "Y" | "N"

1:      'DIRECTION X_T ?'
        "Y" | "N"
        if "N"
                'DIRECTION X_B ?'
                "Y" | "N"
                IF "N"
                      'DIRECTION Z ?'
                      "Y" | "N"
                      if "N"
                            goto 1

        'HELP?'
        "Y" | "N"
```

{ The following outputs to the serial port will proceed unless interrupted by an input to the serial port. An input of E will terminate the help procedure. }

```
        if "Y"
                'A = A/D HEX'
                'B = A/D BIN'
                'C = COEF'
                'D = ARM DOWN'
                'E = EXIT'
                'F = A/D DEC'
                'H = HELP'
                'O = OVERRIDE'
                'R = RESET'
                'S = STOP'
                'T = TIME'
                'U = ARM UP'
                'W = WHICH DIRECTION?'

2:      'READY' | 'OVER' | (<Displacement> | nothing)
```

{ The option between Displacement or nothing defaults to nothing until the base of the display is designated by sending an 'A', 'B' or 'F'. }

```
        if 'READY'
                goto 2
```

```
3:      'OVER' | (<Displacement> | nothing)
        if 'OVER'
                goto 3

4:      <Displacement> | nothing
        goto 4
```

{ The above procedures can be interrupted at any time by a serial input, which when a legal command, will cause the command to be executed and then returned to the procedure in operation. The exception is when an 'R' is sent to the device over the serial line. Receipt of 'R' causes a goto 0. The following pseudocode describes the actions when specific commands are sent to the device over the serial line while the above procedure is being executed. }

```
        if "A"
                { Displacement will be sent as ASCII characters representing the
                hexadecimal value. }

        if "B"
                { Displacement will be sent as ASCII characters representing the
                binary value. }

        if "F"
                { Displacement will be sent as ASCII characters representing the
                decimal value. }

        if "O"
                { Take displacement readings when appropriate, even if the arm is
                not in the proper position. }

        if "W"
                'X_T' | 'X_B' | 'Z'

        if "S"
                'STOP'
                { Operation remains in suspension until a character other than S
                is sent to the device }

        if "H"
                { Print out the entire help directory as given above }

        if "C"
                'HIGH' '1'
                '<coefficient>'
                "<carriage return>" | "<capital letter>" | "<number>"
                if "<number>"
                      { replace coefficient with number }
                if "<capital letter>"
                      { Exit this entire procedure }
```

```
'HIGH' '2'
'<coefficient>'
"<carriage return>" | "<capital letter>" | "<number>"
if "<number>"
     { replace coefficient with number }
if "<capital letter>"
     { Exit this entire procedure }

'HIGH' '3'
'<coefficient>'
"<carriage return>" | "<capital letter>" | "<number>"
if "<number>"
     { replace coefficient with number }
if "<capital letter>"
     { Exit this entire procedure }

'HIGH' '4'
'<coefficient>'
"<carriage return>" | "<capital letter>" | "<number>"
if "<number>"
     { replace coefficient with number }
if "<capital letter>"
     { Exit this entire procedure }

'MID' '1'
'<coefficient>'
"<carriage return>" | "<capital letter>" | "<number>"
if "<number>"
     { replace coefficient with number }
if "<capital letter>"
     { Exit this entire procedure }

'MID' '2'
'<coefficient>'
"<carriage return>" | "<capital letter>" | "<number>"
if "<number>"
     { replace coefficient with number }
if "<capital letter>"
     { Exit this entire procedure }

'MID' '3'
'<coefficient>'
"<carriage return>" | "<capital letter>" | "<number>"
if "<number>"
     { replace coefficient with number }
if "<capital letter>"
     { Exit this entire procedure }

'MID' '4'
'<coefficient>'
```

```
"<carriage return>" | "<capital letter>" | "<number>"
if "<number>"
     { replace coefficient with number }
if "<capital letter>"
     { Exit this entire procedure }

'LOW' '1'
'<coefficient>'
"<carriage return>" | "<capital letter>" | "<number>"
if "<number>"
     { replace coefficient with number }
if "<capital letter>"
     { Exit this entire procedure }

'LOW' '2'
'<coefficient>'
"<carriage return>" | "<capital letter>" | "<number>"
if "<number>"
     { replace coefficient with number }
if "<capital letter>"
     { Exit this entire procedure }

'LOW' '3'
'<coefficient>'
"<carriage return>" | "<capital letter>" | "<number>"
if "<number>"
     { replace coefficient with number }
if "<capital letter>"
     { Exit this entire procedure }

'LOW' '4'
'<coefficient>'
"<carriage return>" | "<capital letter>" | "<number>"
if "<number>"
     { replace coefficient with number }
if "<capital letter>"
     { Exit this entire procedure }

'HIGH RANGE'
'<range limit>'
"<carriage return>" | "<capital letter>" | "<number>"
if "<number>"
     { replace range limit with number }
if "<capital letter>"
     { Exit this entire procedure }

'LOW RANGE'
'<range limit>'
"<carriage return>" | "<capital letter>" | "<number>"
if "<number>"
```

```
                ( replace range limit with number )
            if "<capital letter>"
                ( Exit this entire procedure )
            'SAVE?'
            "Y" | "N"
            if "Y"
                ( The entire set of constants replaces the original set )

    if "T"
            'DELAY .1 SEC'
            "<number>

    if "U"
            ( The arm is raised )
            'RAISED'

    if "D"
            ( The arm is dropped )
```

2.2.2 Remote Mode

Remote mode is achieved by the transmission of a "#" character to the device over the serial input at the time of power up or reset. In remote mode, all instructions to the device consist of one binary character, which is not echoed. The device sends two status commands at appropriate times, namely 0 binary, which indicates an error and 10 binary, which indicates that the device is waiting for a command. The following is a concise listing of the communications in remote mode, all characters given as binary values.

TO	FROM	
	0	Error
	1	Arm is down
	2	Arm is up
	3	Arm is between its extreme positions
	4	X_T calibration in use
	5	X_B calibration in use
	6	Z calibration in use
	7	Overrange in the negative direction
	8	Overrange in the positive direction
	9	Within range
	10	Waiting for a command
	11	Linearized displacement value (decimal number)
12		Request for arm position
13		Request for designation of active calibration
14		Request for status of LVDT
15		Request for displacement value
16		Use X_T calibration
17		Use X_B calibration

18	Use Z calibration
19	Raise arm
20	Lower arm
21	Start over remote

Command binary 12 causes READ_ARM to be called and the binary value of 1, 2 or 3 to be returned. Command binary 13 returns a binary 4, 5 or 6 as appropriate. Command binary 14 returns a binary 7, 8, or 9 as appropriate. Command binary 15 causes three procedures to be called in sequence: READ_ARM, READ AND TRANS_LVDT. The value received from the TRANS_LVDT call is then sent out the serial port in ASCII characters of a decimal number. Commands binary 16, 17 or 18 cause the device to use the corresponding calibration coefficients. Command binary 19 calls RAISE while command binary 20 calls DROP. Command binary 21 causes a looping to the beginning of the remote section and the output of a binary 10 to the serial port. This can be a useful technique for testing whether or not the device is operational.

3. SUGGESTIONS FOR FUTURE DEVELOPMENT

As designed, the repeatability of the combined gage head and four-bar mechanism system is around 200 microinches. If the effects of thermal gradients are compensated then the repeatability could be reduced to at least 50 microinches in each direction. In order to compensate for these effects, a thorough study should be conducted to determine the relationship between temperature and thermal growth. Then, by monitoring the temperature at a respective position on the device, a prediction can be made of the change in location of the gage head. In this way the growth of the device with respect to the arm can be virtually eliminated. Another way of reducing the effects of thermal gradients would be to change the mechanical design. For instance, a shorter arm or an arm made out of a material with a coefficient of thermal expansion near zero, such as invar, could be used.

A thorough study should also be conducted on the effectiveness of using this tool setting system to monitor tool wear. So many factors enter into the determination of tool wear that the process is not as straightforward as it would appear. For instance, this system will only supply information on the amount of flank wear of a tool, yet under certain cutting conditions a tool will fail due to excessive crater wear. In addition, under certain conditions, a tool develops a built-up edge on its cutting surface called bue. The presence of bue will yield erroneous results when using this method of tool setting. An extensive study of tool wear, however, should lead to results such as a relationship between crater wear and flank wear for different cutting conditions.

Since the machine tool controller contains the necessary microprocessor, there could easily be an integration of machine tool and tool setting station controllers. However, some additional hardware components will be necessary for such integration.

IX. MICROMANIPULATOR

The micromanipulator, located between the gripper and wrist of the robot, assists in the placing of parts into the collet of the lathe by touch sensation. Placement of a part into a collet consists of search and insertion routines which take advantage of the micromanipulator's five degrees of freedom.

1. DESIGN OF THE MICROMANIPULATOR

1.1 Overview

The micromanipulator is a light-weight, compact, hydraulically driven, fine positioning device with five degrees of freedom. Its motions consist of two mutually orthogonal rotational degrees of freedom and three mutually orthogonal translational degrees of freedom. This design was selected to allow the end effector to insert parts in a collet, considering the poor position repeatability of the robot and the small clearances between part and collet.

The hardware consists of an Intel 86/30 single board computer with an 8087 math coprocessor on board. In addition, there are five A/D (analog-to-digital) ports and six D/A ports along with an RS232 serial port for communication. This is a versatile microprocessor board which interfaces well with many electronic devices.

Two programs, written in PL/M, comprise the software. Since this controller was to perform one function (control of the micromanipulator), no operating system was deemed necessary. This high-level language was chosen because it can descend to the individual bit level when necessary and has all of the advantages of a structured high-level language.

1.2 Control Architecture

The control consists of a software servo-loop operating at 50 hertz which constantly attempts to position the micromanipulator at a desired location and orientation in space. The position of each of its degrees of freedom relative to its base is known from the readings of five position sensors which are read during each cycle of the servo-loop. With this system, the device will respond within 20 milliseconds to any force causing a change of position of any of the degrees of freedom or to a change in the desired location of any of these degrees of freedom. All of the various move and search routines involve modifications of the parameters in this servo-loop.

1.3 Mechanical Components

The micromanipulator's mechanical system consists of four principle parts: 1) a base structure which attaches to the robot, 2) a midstructure which moves in two degrees of freedom with respect to the base structure, 3) an end structure which moves with respect to the midstructure in two degrees of freedom, 4) and a gripper mounting plate having one large translational degree of freedom with respect to the end structure. Each structure is comprised mainly of aluminum and in some cases is made up of several components. An isometric view and a top view of the micromanipulator system is shown in Figures IX.1 and IX.2, respectively. The large translational degree of freedom will be referred to as the slider and the other four as the first four degrees of freedom.

The base and midstructure assembly form the vertical and yaw motions. Two opposing pistons on the midstructure are able to slide in the vertical, or z-axis direction within cylinders in the base structure. These opposing pistons also form the yaw axle about which the midstructure rotates as shown in Figure IX.3. The yaw motion is driven by two opposing pistons that slide in cylinders machined integral with the midstructure as shown in Figures IX.4 and IX.5. The cylinders are lined with hardened steel sleeves which prevent the steel pistons from scoring the softer aluminum cylinder walls. To prevent sliding between the piston and base structure during a vertical motion, a special ball and trough interface was developed. The steel ball is sandwiched between the trough shape of the piston end and the trough shape of the base structure. This configuration generates rolling contact even if two degrees of freedom are actuated at one time. The radii of the ball and troughs, along with the material and its hardness were chosen so that the maximum contact stress did not exceed 75% of the yield strength of the material.

The midstructure is also assembled with the end structure and together they form the forward and roll motions of the micromanipulator. A large piston attached to the end structure extends into a bore in the midstructure providing the forward, or x-axis motion. The piston is supported at either end by low friction sleeve bearings. This piston also serves as the roll axle as shown in Figures IX.3 and IX.4. The rolling action is driven by the same type of piston/ball/trough configuration as used for the yaw motion. These pistons are located in bores in the midstructure.

The remaining degree of freedom, or the slider, provides linear motion along the y-axis. The slider is driven by a linear motion piston and cylinder assembly which is attached to the end structure and the gripper mounting plate. Two sets of crossed roller bearings allow the gripper plate to slide relative to the end structure.

The micromanipulator design results in a device that weighs about five pounds and is 2.5 inches thick by 5 inches square. The motion consists of two translational axes with +/- 0.125 inch of travel, two rotational axes with +/- 4 degrees of rotation, and one large translational degree of freedom with +/- 1.0 inch of travel.

Position feedback of the first four degrees of freedom is obtained from inductive proximity sensors as shown in Figures IX.2 through IX.5. The sensors detect displacements along the axis of the probe and are not affected by motion perpendicular to this axis. Rotational motion is sensed as small linear

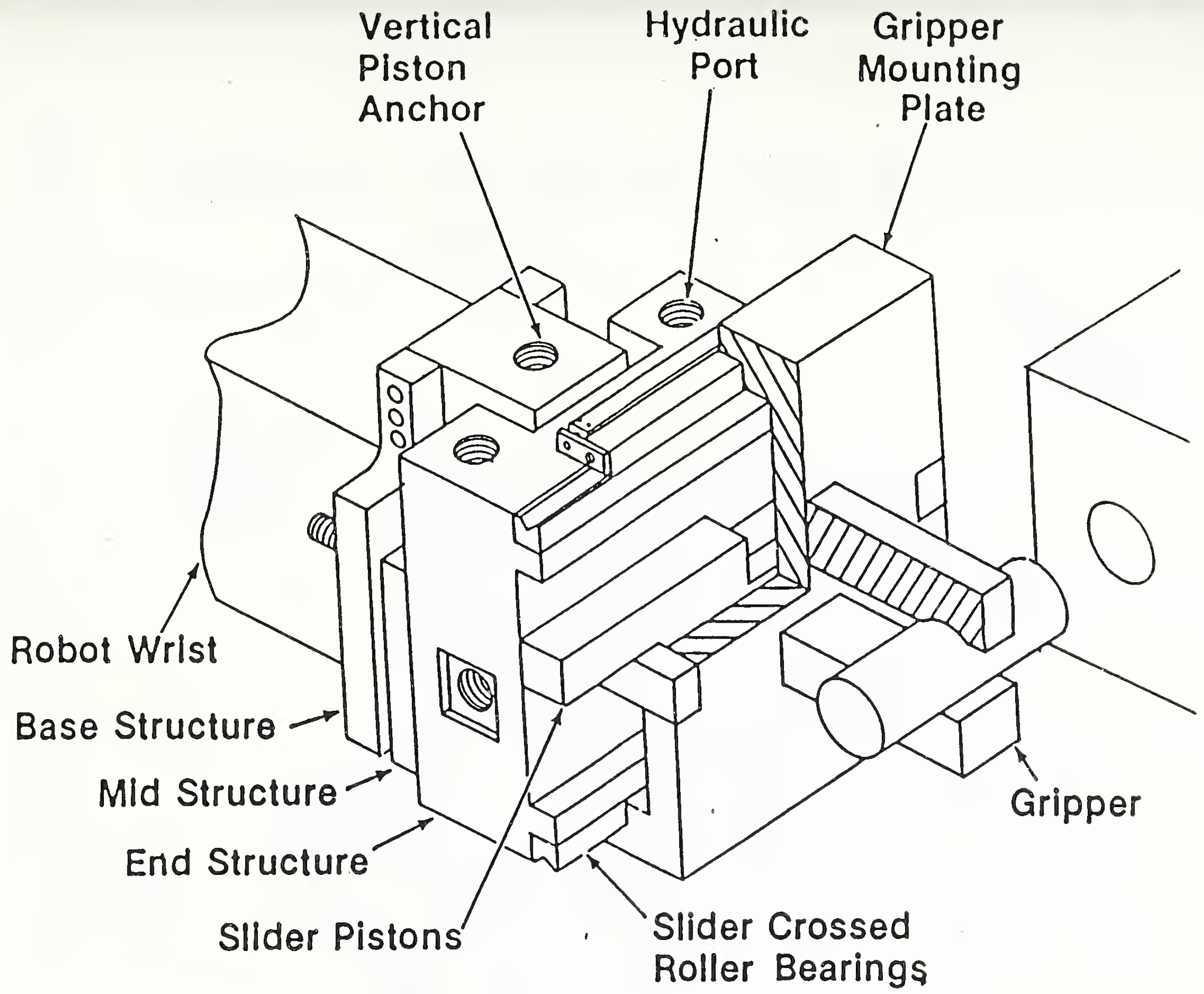


FIGURE IX.1. MICROMANIPULATOR STRUCTURE

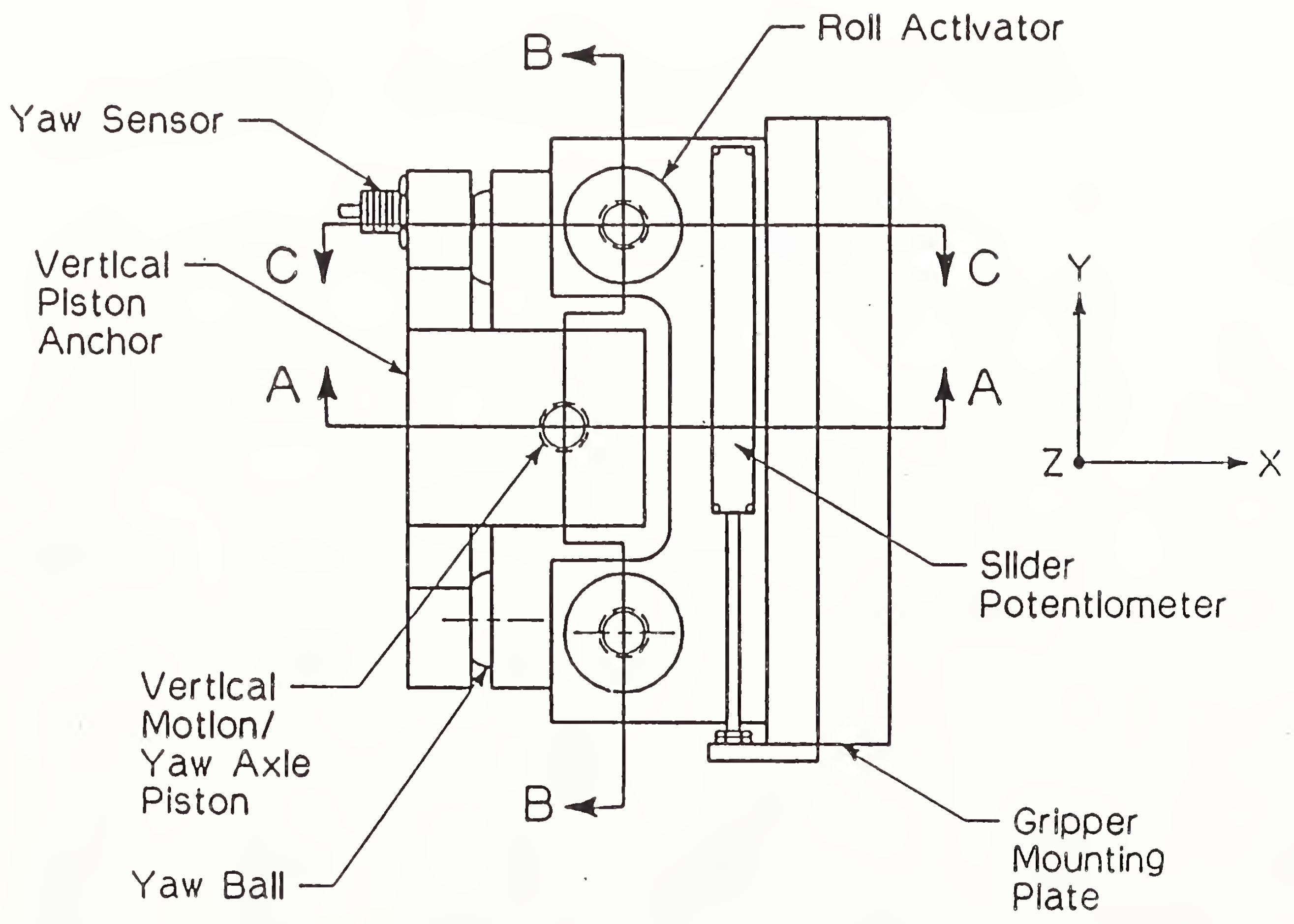


FIGURE IX.2. TOP VIEW OF MICROMANIPULATOR

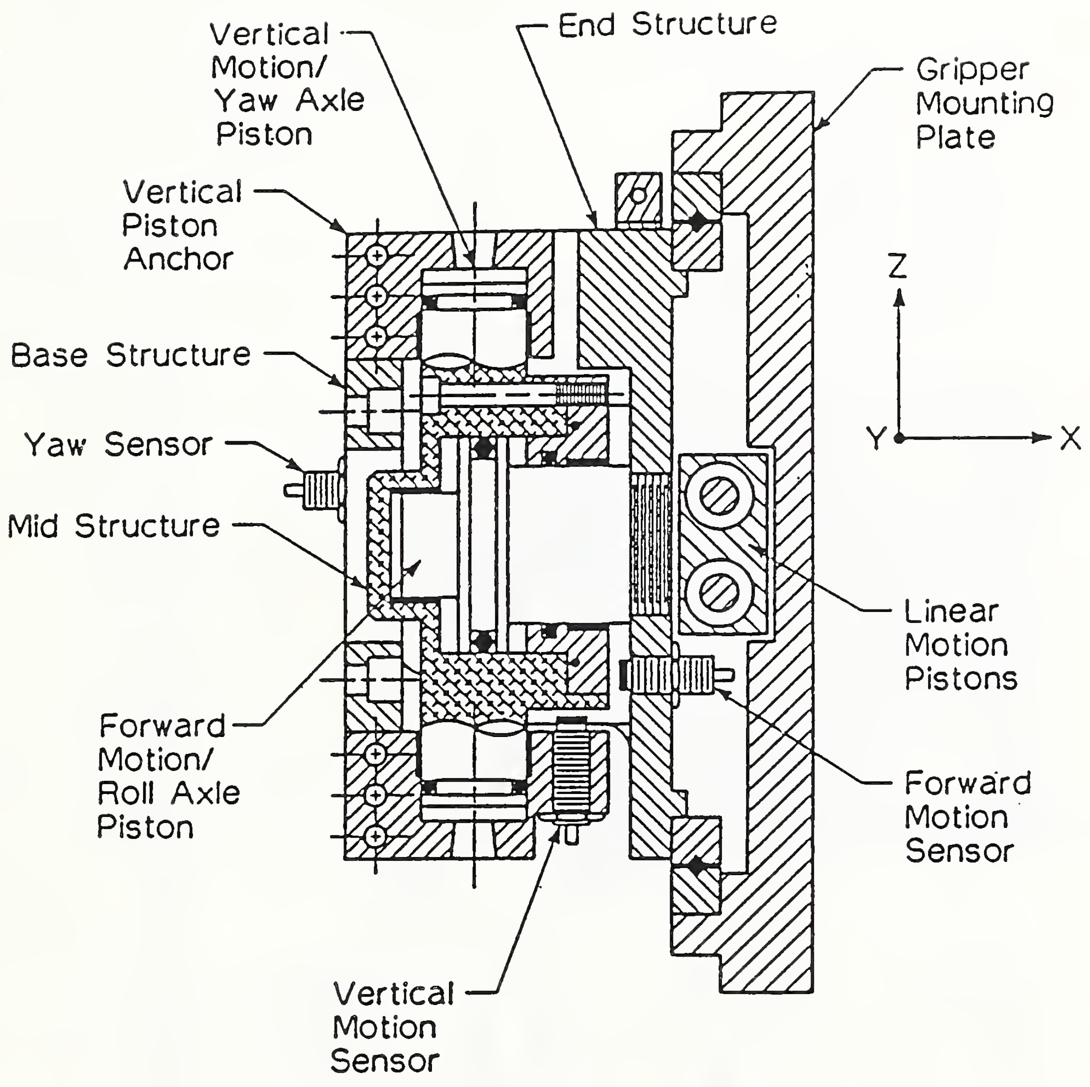


FIGURE IX.3. SECTION AA OF MICROMANIPULATOR

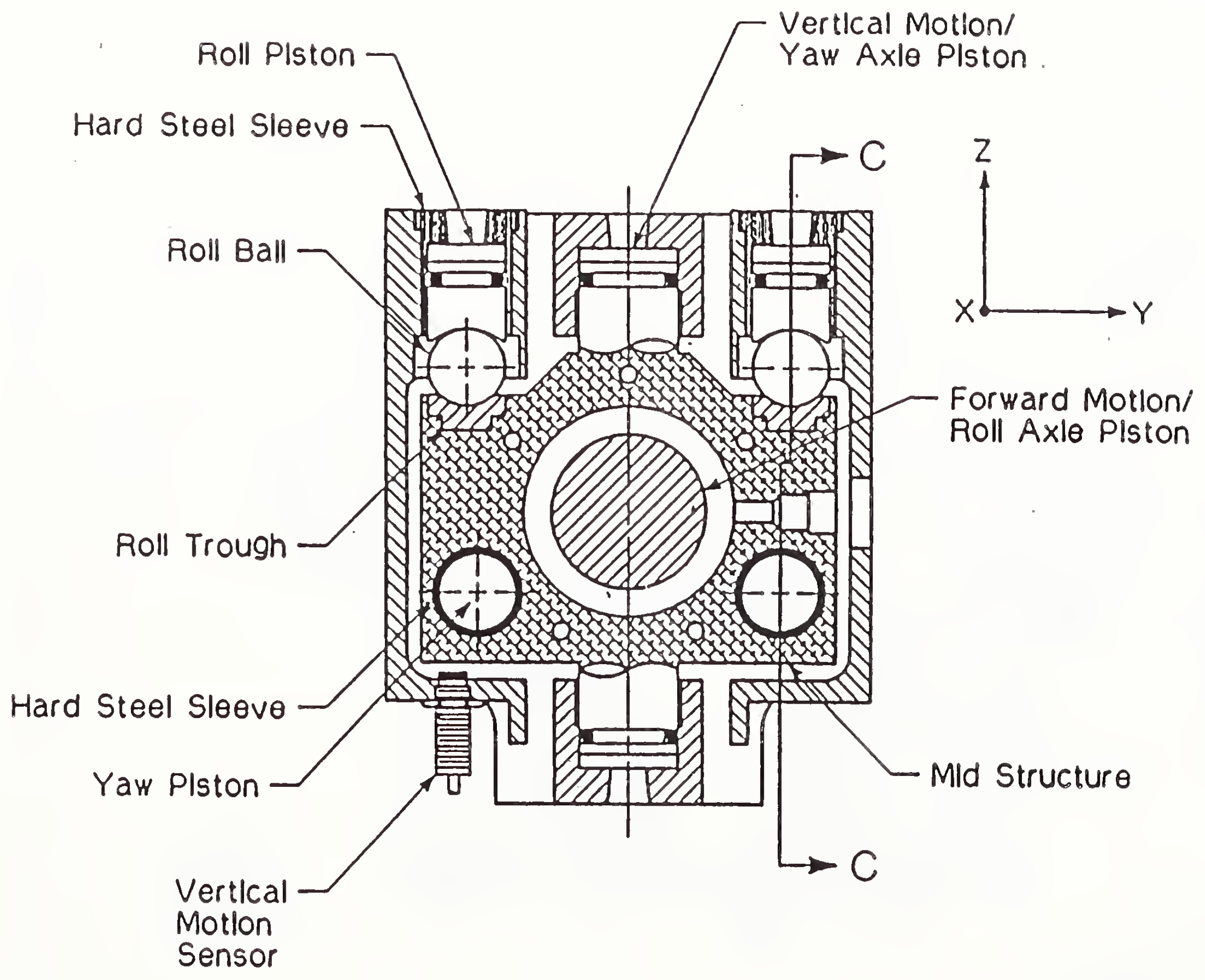


FIGURE IX.4. SECTION BB OF MICROMANIPULATOR

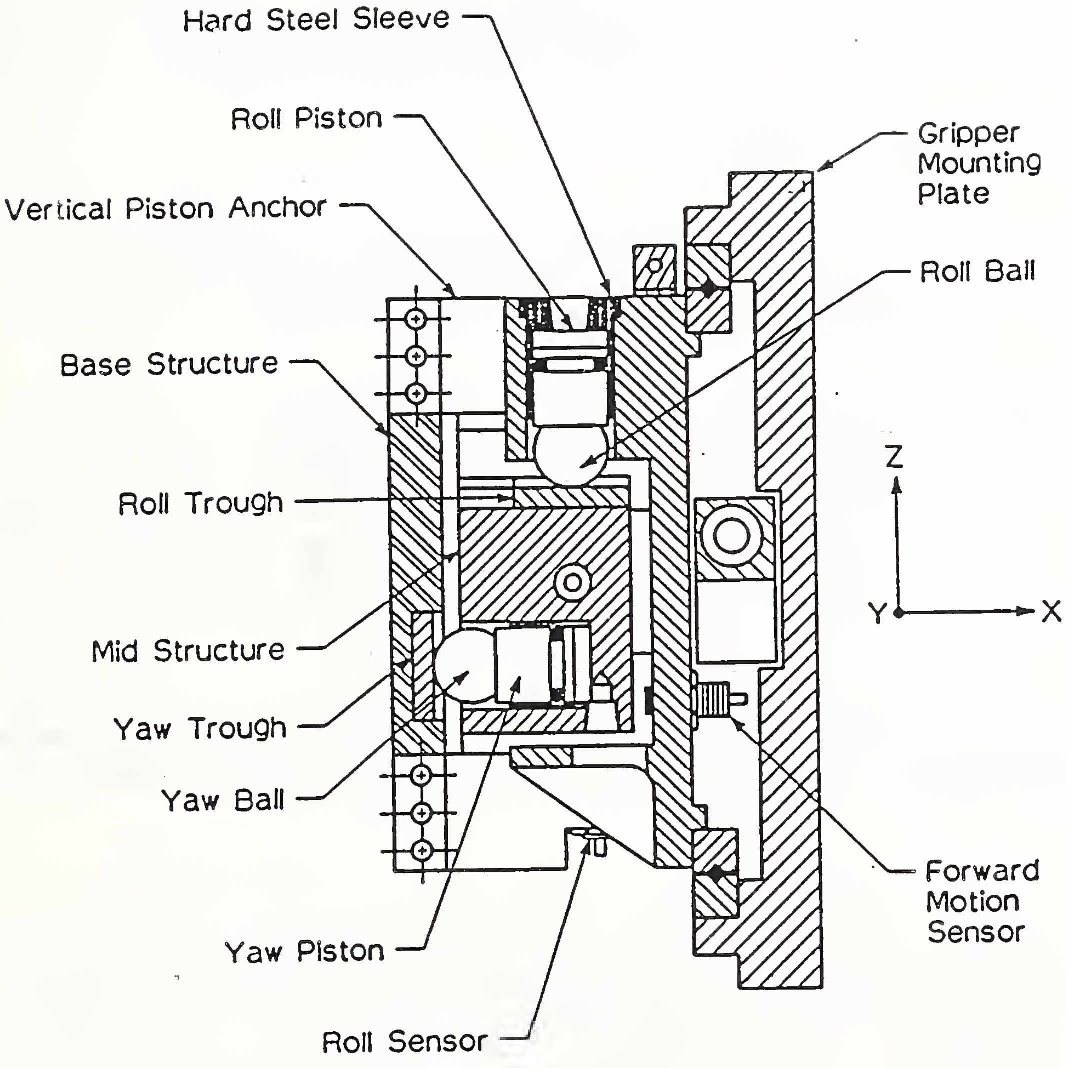


FIGURE IX.5. SECTION CC OF MICROMANIPULATOR

translations at a known distance from the axis of rotation. The slider position is determined through the use of a linear potentiometer. Each actuator in the system is driven by hydraulic fluid. The flow of hydraulic fluid is controlled by a separate servo-valve for each degree of freedom. Each valve can operate at a maximum pressure of 3000 psi. and can produce a differential pressure output of up to +/- 80% of supply pressure. The valves are mounted on manifolds on the robot wrist so that the length of hose between the valves and the micromanipulator is kept to a minimum. In addition to these valves, a hydraulic control valve is used to turn off the pressure to the first four degrees of freedom. This provides a means for "relaxing" the micromanipulator.

1.4 Electronic Components

The electronic hardware components for the micromanipulator consist of a 16-bit 8086 microprocessor working in conjunction with an 80-bit floating-point math coprocessor. The program is contained on electrically programmable read-only-memory capable of containing 64 Kbytes of program. There is also 128 Kbytes of random-access-memory available.

There are five analog-to-digital input ports which convert the 0 to 10 volt signals, put out by the sensors, into digital values which are read by the microprocessor as position feedback of the manipulator axes. There are six digital-to-analog (D/A) output ports, five of which are used to convert digital signals from the microprocessor into control voltages for each of the proportional valves, in order to control motion in the five directions. The sixth D/A port is for sending the signal to the on/off hydraulic valve in order to relax four of the degrees-of-freedom.

An RS232C serial I/O port is used to communicate with the outer world. This port is operated at 9600 baud, and communicates either with another computer, during automated operation of the system, or to a dumb terminal, when testing the micromanipulator.

1.5 Software Components

The software for the micromanipulator consists of two programs, written in PL/M and linked together to form one module. One of the programs consists of the initialization parameters for the software, such as the calibration factors on each of the position sensors, the servo constants, etc. These parameters exist in a separate program, in order to perform quick changes to a short program, which can be compiled rapidly and linked with the other much larger program.

The larger of the two programs consists of thirty-two procedures, a number of which can be called by a single ASCII character sent to the microprocessor by means of the serial port. During any operation, a hardware clock interrupts the program at a fixed frequency, the frequency currently being 50 hertz. This interrupt causes the servo-loop to update the values of the sensor readings and to send to the hydraulic valves newly computed values, based on these new readings and any new commands that may have come in by means of the serial port. Among the procedures available in automated mode are procedures to move all five pistons to particular positions, to relax four of the five pistons, to move one of the pistons to a particular position, to move in any direction with a predetermined amount of force, and to perform combinations of these operations, such as conducting a search routine in order to locate a collet hole or to withdraw a part from a collet.

Local mode includes an exercise routine as well as test and diagnostic routines. A larger version of the program contained test routines for obtaining optimum servo constants. Once obtained, these constants became part of the initialization routine and the smaller version is used, in order to facilitate software maintenance. A list of all of the procedures in the operating program along with a short description of the function of each procedure follows:

ALEX_ALGO:	Sets up three constants for the servo-loop for a particular degree of freedom. This procedure must be run five times to set up all of the servo-loop constants.
ALLIGN:	Alter the two rotations while testing for maximum travel of the slider to improve the alignment of the slider. Retain, for future use, the rotational positions that were present when maximum slider motion was achieved.
AT_EASE:	Call PRESSURE(off).
ATTENTION:	Call PRESSURE(on).
CENTER:	Position any of the first four degrees of freedom to its center position and hold until further instructions.
CENTER_ALL:	Position all of the first four degrees of freedoms to their center position and hold the positions until further instructions.
COMPLETION:	Wait for two cycles of the servo-loop.
CORRECTION:	Move the device through a pattern of up to 121 locations. At each position call ED. If successful in ED or stuck in ED terminate the operation and report the results.
DISTANCE:	Move any of the first four degrees of freedom between its extreme positions while trying to make a small move with the slider.
DONE:	This is an interrupt procedure that is activated by a hardware clock (presently every 20 milliseconds). Done calls a procedure EXECUTION.

ED:	Try to move the slider a designated distance in a designated direction. If successful, report the success. If unsuccessful, try to determine if the slider is stuck in its present position and report that condition if it exists.
EXECUTION:	Obtains the current position for all active degrees of freedom. This involves getting the A/D readings and converting these readings into positions in accordance with linearization equations. Based on these positions and the desired positions and the previous conditions, it calculates the new values necessary for the D/A converters and puts them out to the converters in order to position the device.
GET_A_D:	Obtain the position of any degree of freedom.
HOLD:	Set the force for the first four degrees of freedom to zero and wait the number of seconds given in the transfer variable.
INIT_PIC:	Initialize the programmable interrupt controller.
LOCK:	Set the new position for all five degrees of freedom to the corresponding present position. Call ATTENTION.
MOVE:	Move any of the degrees of freedom in either direction with a designated force.
MOVE_WITH:	Move the first two degrees of freedom between their extreme positions (chatter) while trying to make a small move with the slider.
NEW_SET:	Set the driving force on the first four degrees of freedom to zero. Call AT_EASE and call RESET.
PRESSURE:	Controls the pressure to the first four degrees of freedom valves.
PROGRAM_ALT:	Position the slider to a position 5% of the way to its negative limit.
PROGRAM_ALTERNATE:	Position the first four degrees of freedom to their center position and the slider to a position 5% of the way from its negative limit less any designated offset.
PROGRAM_POS:	Position the slider to a position 5% of the way to its positive limit.

PROGRAM_POSITION: Position the first four degrees of freedom to their center position and the slider to a position 5% of the way to its positive limit less any designated offset.

RESET: Record the present position of the first four degrees of freedom.

REPORT: Send the error result of a procedure to the serial port.

SO: Local mode operation (using a dumb crt).

SEARCH: Completely untested procedure. Repeatedly calls CORRECTION while slightly modifying the translational degrees of freedom of the first four in order to find an optimum offset in the base position from which to operate the future CORRECTION procedures.

SHOVE: Do a STOP_PT_GRAD on the slider with 5% of its maximum force.

STATUS: Send a carriage return and a line feed followed by the five degrees of freedom drive force values and current positions to the serial port. Follow that with the two preferred angle positions as determined by ALLIGN.

STOP_PT: Move any degree of freedom with any prescribed force and obtain the end position of the move.

STOP_PT_GRAD: Move any degree of freedom with any prescribed force until the first degree of freedom (horizontal rotation) meets a preset resistance to maintenance of its position. Obtain the end position of the move.

TIMER: Load the hardware clock. This determines the period between updates of the servo-loop.

2. OPERATION OF THE MICROMANIPULATOR

Upon powering up the micromanipulator, the program causes a reading to be taken of the current position of all degrees of freedom and then inserts these values into the servo-loop in such a way as to lock these positions until receiving a command to do otherwise. A character is then sent out of the serial port and the microprocessor waits for one of two characters to be sent to it over the serial port. One of the characters causes remote mode to be activated, while the other causes local mode to be activated.

2.1 Local Mode

In local mode, communication between the dumb terminal and the microprocessor is carried on in English, primarily with the microprocessor asking questions in English and the operator at the terminal responding with "y" or "n". In addition, the microprocessor can occasionally request the input of parameter values, in which case an ASCII representation of the number is accepted by the microprocessor. In this mode, everything that the operator types at the terminal is echoed to the screen in order to facilitate human communication. Remote mode can also be simulated in local mode when needed for diagnosis, but, in general, the operator can accomplish any thing desired in the more user-friendly local mode.

The following pseudocode displays the operation under local mode. All messages sent by the detector are enclosed in single quotes and responses in double quotes with items within square brackets, "[ ]", being optional. Items between which choices must be made are separated by the vertical bar, "|", while dummy values are shown as "< >", and non-operable comments are shown within brackets, "( )". Numbers followed by a ":" are labels.

```
1:      'u'

( Any of the following unsolicited commands can now be given.  Discussion of
each will follow the list. )

    "#"     Obtain option to relinquish control.

    "a"     Align axes using positive direction.

    "d"     Activate first four degrees of freedom (fifth degree is always
            active).

    "e"     Exercise desired degrees of freedom at desired force.  0 force
            turns off that degree's exercise.

    "l"     Hold the micromanipulator at its current position.

    "h"     Delay following a zero pressure set

    "p"     Check push force.  Alter if desired.

    "r"     Alter remember function.  Defaults to on when powered up.

    "s"     Obtain status printout.

    "t"     Check touch force.  Alter if desired.

    "o"     Check if dropped part function is on.  Alter if desired.
```

    SPACE   Initiate positioning queries.

{ pseudocode continues with commands given in the order that they occur in the program but which could be given in any order }

```
       if "e"
              'Exercise motors?'
              if "Y"
                   'Period between moves in seconds'
                   "< >"
2:                 'Key force # 1'
                   "< >"
                   if "< > greater than max force for # 1
                        'Too High !!!    Higher than <max force # 1>'
                        goto 2
3:                 'Key force # 2'
                   "< >"
                   if "< > greater than max force for # 2
                        'Too High !!!    Higher than <max force # 2>'
                        goto 3
4:                 'Key force # 3'
                   "< >"
                   if "< > greater than max force for # 3
                        'Too High !!!    Higher than <max force # 3>'
                        goto 4
5:                 'Key force # 4'
                   "< >"
                   if "< > greater than max force for # 4
                        'Too High !!!    Higher than <max force # 4>'
                        goto 5
```

{ The device now proceeds to move each degree of freedom for which a force greater than 0 was entered. Degrees of freedom for which a 0 force was entered, are maintained at their present location. It applies the specified force (1 to 2000) to the appropriate degree of freedom for the designated period. At the end of the period, the device sends to the CRT the following: }

```
  '<limit 1>      <limit 2>      <limit 3>      <limit 4>      <limit 5>'
  '<read 1>[**]  <read 2 >[**]  <read 3 >[**]  <read 4 >[**]  <read 5 >[**]'
```

{ The limit value is a position value that the degree of freedom should be able to reach when the device is well exercised and operating properly. The '**' appear appropriately whenever the read position has not attained the corresponding limit position. The limits and readings are only printed for the degrees of freedom where the input force value was greater than 0. After each period, the force value is reversed and each degree of freedom moves in the reverse direction and the reverse limits and readings are given. The reverse limits may or may not be the same magnitude but are of opposite sign. The appropriate limit values for both directions have been obtained experimentally.

The exercising continues until a key is pressed on the CRT at which time the driving force for all degrees of freedom is set to 0 and position holding of all degrees of freedom is activated. }

```
        if "l"
                { call LOCK }

        if "r"
                'Remember?'
                "Y" | "N"
```

{ A "Y" turns on the function that causes the device to use as the first position of its search routine, the position last obtained in a successful search in the same direction. }

```
                'u'

        if "s"
                '<Drv 1>,<Drv 2>,<Drv 3>,<Drv 4>,<Drv 5>   Drive value'
                '<Pos 1>,<Pos 2>,<Pos 3>,<Pos 4>,<Pos 5>   Position'
                '<Dsr 1>,<Dsr 2>                           Desired positions'
```

{ Drv is the servo driving force value. If the position is stable at the desired location, the Drv value would be 0. Pos is the current position of the particular degree of freedom and Dsr is the value of the two angular degrees of freedom which will be served during a search routine.}

```
        if "a"
                { call ALLIGN in the positive direction }

        if "t"
                '< touch force >'
                'Set touch force?'
                "Y" | "N"
                if "Y"
                      'Key in force (decimal)'
                      "< force >"
                'u'
```

{ When the slider pushes against an external object, a force tending to rotate the first degree of freedom accompanies the slider push against the object. Whenever this first degree of freedom force exceeds the touch force during motion of the slider, it is taken as the signal that the slider has touched an external object. }

```
        if "o"
                'Dropped part function is on' |
                'Dropped part function is NOT on'
                'Change function?'
                "Y" | "N"
                'u'
```

```
{ The dropped part function attempts to indicate that a part has been dropped.
      In a search routine, if the slider is able to move beyond a certain
      point, it is taken as a successful search.  If the dropped part function
      is on, then if the slider has moved further than a preset limit, it is
      not taken as a successful search but an indication that there was no
      part that prevented the motion.  }

      if "p"
             '< push force >'
             'Set push force?'
             "Y" | "N"
             if "Y"
                  'Key in force (integer)'
                  "< force >"
             'u'

{ The push force is the force applied to the slider when it is desired to move
the slider until it touches an external object. }

      if "h"
             'Set delay time?'
             "Y" | "N"
             if "Y"
                  'Key in delay (integer in 0.1 seconds)'
                  "< delay >"

{ Delay is the time interval that is waited after the force of the first four
degrees of freedom are set to 0 in the HOLD procedure. }

      if "d"
             { call ATTENTION }
             'b'
             'b'
             'r'

6:    if "space"     { the character, not the word }
             'Programming position?'
             "Y" | "N"
             if "Y"
                  'b'
                  'r'
             if "N"
                  'Alternate programming position?'
                  "Y" | "N"
                  if "Y"
                       'b'
                       'r'
                  if "N"
                       'Aligned programming position?'
```

```
            "Y" | "N"
            if "Y"
                'b'
                'r'
```

{ The programming position with the two rotations set to the values obtained from align. }

```
            if "N"
                'Aligned alternate programming position?'
                "Y" | "N"
                if "Y"
                    'b'
                    'r'
```

{ The alternate programming position with the two rotations set to the values obtained from align. }

```
    'Repeat positioning?'
    "Y" | "N"
    if "Y"
        goto 6

if "#"
    'Maintain CRT control?'
    if "Y"
        goto 1
    if "N"
        goto remote mode

goto 1
```

2.2 Remote Mode

In remote mode, all instructions to the microprocessor consist of one ASCII character, which is not echoed by the microprocessor. The microprocessor sends status characters to the automated operator such as "r" or "b", indicating ready or busy, respectively. When the microprocessor receives an illegal or inappropriate command, it responds with the character "n" and awaits the resubmission of the command. Whenever a problem occurs, such as the inability of the microprocessor to perform the task commanded, the microprocessor issues an "e" to the serial port and, when it receives an "i" at the serial port, it issues a coded character that indicates the type of error encountered. The following table is a concise listing of the communications in remote mode:

TO	BOTH	FROM	
		'u'	Waiting for initial command
CTR A			Start in local mode
'#'			Start in remote mode

	'a'		Acknowledge message
		'b'	Busy
'c'			Clear
'#'			Restart in remote mode
'd'			Lock micro where it is
		'e'	Error [followed by error code (ASCII 0 - 9)]
'F'			Move to aligned home position
'g'			Move slide - till touch
'h'			Move to home position
'H'			Move to home position - instructed amount
	'i'		Inquiry
'j'			Move to alternate home position
'J'			Move to alternate home position + instructed amount
'k'			Move slide + till touch
'L'			Move to aligned alternate home position
'm'			Search for hole [followed by strategy (ASCII 0 - 9)]
	'n'		Not understood
'p'			Push part in collet - direction
'P'			Push part in collet - direction with extra testing
'q'			Push part in collet + direction
'Q'			Push part in collet + direction with extra testing
		'r'	Ready
'r'			Relax manipulator
'R'			Set zero force on first 4 degrees of freedom
's'			Status request
't'			Relax and record for next position
'u'			Push hard in - direction
		'u'	Waiting for command
'v'			Push hard in + direction
'U'			Push hard in - direction with Jitter
'V'			Push hard in + direction with Jitter
'w'			Move slide to home position
'x'			Move slide to alternate home position
'y'			Align angles for insertion in positive direction
'Y'			Align angles for insertion in negative direction
'z'			Emergency stop (Lock micro where it is)

Errors will be ASCII digits 0 to 9

1	Micro didn't move on push (into collet)
7	No part (micro couldn't) touch off

A short elaboration of commands will now be given:

The first 'u' is a character sent on power up or reset to indicate that the system is working. There are only two acceptable responses to this 'u', a control-A or a '#'. The control-A causes the device to go into local mode while the '#' causes the device to go into remote mode. Upon receiving a '#', then device sends an 'r', indicating a readiness to receive a command.

If a 'c' is sent, the device responds with a 'b' then resets and sends a 'u'.

If the device receives a 'd', it responds with a 'b', calls LOCK and when finished, it sends an 'r'.

When an 'F' is received, it responds with a 'b', sets a flag instructing the use of the alignment positions for the first two degrees of freedom, calls PROGRAM_POSITION with no offset, then calls COMPLETION and sends an r.

Upon receiving a 'g', STOP_PT_GRAD is called with the move in a negative direction. If a touch is achieved, the process ends with the sending of an 'r'. If a touch is not achieved, the process is repeated up to four additional times. If a touch is achieved at any try, the process is stopped and an 'r' is sent. If no touch is achieved, REPORT is called with an error signal of '1'. The error signal can be modified to accommodate the station controller.

An 'h' causes a 'b' to be sent and PROGRAM_POSITION to be called followed by COMPLETION and an 'r'.

When the device receives an 'H' it sends an 'i' and waits to receive a number in integer, decimal, or exponential form; each character of the number being in ASCII form. The number must be terminated with a carriage return. Upon receiving this number, the device sends a 'b' and PROGRAM_POSITION is called with an offset corresponding to the number received. This is followed by COMPLETION and the sending of an 'r'.

If an 'i' is received at this time an 'r' is sent and nothing happens until a character other than 'i' is sent. The usage of 'i' is restricted to the time when an error signal is sent and the error designation is requested.

A 'j' causes a 'b' to be sent and PROGRAM_ALTERNATE to be called followed by COMPLETION and an 'r'.

When the device receives a 'J' it sends an 'i' and waits to receive a number in integer, decimal, or exponential form; each character of the number being in ASCII form. The number must be terminated with a carriage return. Upon receiving this number, the device sends a 'b' and PROGRAM_ALTERNATE is called with an offset corresponding to the number received. This is followed by COMPLETION and the sending of an 'r'.

Upon receiving a 'k', STOP_PT_GRAD is called with the move in a positive direction. If a touch is achieved, the process ends with the sending of an 'r'. If a touch is not achieved, the process is repeated up to four additional times. If a touch is achieved at any try, the processes is stopped and an 'r' is sent. If no touch is achieved, REPORT is called with an error signal of '1'. The error signal can be modified to accommodate the station controller.

When an 'L' is received, it responds with a 'b', sets a flag instructing the use of the alignment positions for the first two degrees of freedom, calls PROGRAM_ALTERNATE with no offset, then calls COMPLETION and sends an r.

An 'm' starts a search after offsetting the rotational axis specified amounts. There are ten possible preset offsets (0 to 9). Upon receiving the 'm', the device sends an 'i' and waits for a digit in ASCII of 0 to 9. Receipt of anything but the permitted digit causes the device to send an 'n' and wait for the proper digit. Upon receiving an improper response for ten times, the device sends an 'r' to indicate that it is now looking for a new command. Upon receiving the acceptable digit, the device presets the rotational degrees of freedom by the amount specified in the called for case and proceeds to call CORRECTION and do the search in the normal method. The amount of motion that the slider is able to perform is used as a measure of whether the search has been successful. If unsuccessful, the error signal is given, otherwise an 'r' is returned.

A 'p' causes the device to send a 'b' and start a search in the negative direction by calling CORRECTION as in the 'm' routine. The only difference is that the rotational degrees of freedom are not preset to a different position.

The 'P' routine operates exactly as 'p' with one additional test. If it is determined that the slider has not moved sufficiently to indicate that the search was successful, an additional test is performed in order to determine whether or not all motion of the slider is impeded, indicating that the part is stuck. In this case, it is considered a success.

Receipt of a 'q' causes the exact same actions as 'p', except that the motions of the slider are in the reverse direction.

The receipt of a 'Q' causes the same actions as 'P', with the difference that the slider motions are in a reverse direction.

When an 'r' is received, a 'b' is sent followed by the setting of a zero driving force on the first four degrees of freedom. AT_EASE is then called which sends another 'b', cuts off the hydraulic pressure to the first four degrees of freedom, then waits two seconds before returning an 'r'.

The receipt of an 'R' causes a 'b' to be sent followed by a call to HOLD which sets the driving force on the first four degrees of freedom to

zero and waits one tenth of a second before returning an 'r'.
The receipt of 's' causes STATUS to be called which is then followed by the sending of an 'r'.

NEW_SET is called upon the receipt of a 't'. This is followed by the sending of an 'r'.

The receipt of a 'u' causes the slider to be moved in the negative direction with a force equal to the previously established maximum force for this degree of freedom. This is followed by the sending of an 'r'.

A 'v' causes the same actions as a 'u', but the motion is in the positive direction.

The receipt of a 'U' causes MOVE_WITH to be called in the negative direction. This results in the same action as a 'u' with the rotational degrees of freedom swinging between their extremes in order to prevent sticking of a part.

The receipt of a 'V' causes the exact same action as the 'U' but the slider moves in the positive direction.

A 'w' causes PROGRAM_POS to be called, followed by the calling of COMPLETION after which an 'r' is sent.

The receipt of an 'x' causes PROGRAM_ALT to be called, followed by the calling of COMPLETION after which an 'r' is sent.

When a 'y' is received, the device calls ALLIGN with the slider motion being in a positive direction. If during this procedure the slider is unable to move a prescribed minimum amount in that direction, an error signal is given. If the motion is achieved, an 'r' is sent. This procedure is valuable following a successful search, in order to align the rotational degrees of freedom more closely with the desired direction of slider travel.

The receipt of a 'Y' causes the identical operation as a 'y' with the exception that the slider motion is in a negative direction.

The receipt of an 'x' causes the present five degrees of freedom to be servod at their present position and insures that the pressure is on to all of the degrees of freedom. This is accomplished by calling LOCK.

3. SUGGESTIONS FOR FUTURE DEVELOPMENT

The micromanipulator, as presently designed and built, has no method of measuring the force with which it is attempting to move any of its degrees of freedom. This forces it to try to get a handle on the force by a roundabout method, which is only moderately successful. If force sensors existed, many of the procedures which are now performed either with difficulty or with an unacceptable rate of failure, could be performed easily with practically a zero failure rate.

It is highly likely that position sensors which could be more easily adjusted to measure the entire motion of each of the degrees of freedom and would maintain this adjustment could be found. Calibrating the position sensor output against positions of the micromanipulator is a difficult and time-consuming task and it would be advantageous to require that this only be necessary once.

It might be advantageous to have a learn mode in the microprocessor. Although certain positions resulting from particular operations can be saved and used in the next execution of the same operation, it might be useful for the microprocessor to have the ability to retain a complete series of successful moves in which a desired result was obtained. In order to accomplish this learning task, it would be desirable to have some nonvolatile random-access memory in order to be able to retain that which was learned.

X. MALFUNCTION DETECTOR

Automated untended operation of equipment, especially equipment with a potential safety hazard, requires some form of automated monitoring of the equipment to insure that malfunctions will not result in expensive or dangerous consequences. The malfunction detector is an instrument designed to detect most occurrences of improper operation at an automated workstation. This is accomplished by monitoring the vibration caused by machine operations and comparing the vibration level with predetermined upper and lower limits of acceptable vibration. Obtaining a vibration level outside of the allowable limit, a signal is given, received by another computer and, upon inquiry from the controlling computer, a code is given to designate which malfunction occurred. This provides a means of stopping an operation in the event of tool breakage caused b y premature tool failure, unproper tool loading, execution of the wrong NC program, etc.

1. DESIGN OF THE MALFUNCTION DETECTOR

1.1 Overview

The malfunction detector uses two accelerometers mounted on the tool holding turret assemblies of the turning center. The mounting orientation chosen is optimal for the smallest signal level differences of the various cutting operations.

The electronics hardware consists of two in-house printed-circuit boards. One board provides signal conditioning, RMS, Log, and analog-to-digital (A/D) conversion; the other is a computer that utilizes a math coprocessor and battery-backed read-only memory for storing operational tolerances.

The software consists of one program, written in PL/M, with no operating system. Since this controller was to perform one task, no operating system was deemed necessary. The PL/M language was chosen because it can address the individual bit-level when necessary and has all of the advantages of a structured high-level language. This language was designed to be used primarily with Intel microprocessors, the type of microprocessors used with this device.

1.2 Control Architecture

The controller of the malfunction detector consists of a system that receives signals from the machine tool each time another process is begun and ended. By keeping track of these signals, the microprocessor can know the characteristics of the process in operation at any particular time. The microprocessor compares the expected characteristics of the current process with the signals received from the accelerometers. A decision as to whether the signal levels are within the tolerance set for that process determines whether or not a malfunction signal is to be given.

1.3 Mechanical Components

The mechanical components of the malfunction detector include two accelerometers mounted on the tool holding turret assemblies of the turning center. The accelerometers are Endevco Model 7701.50. These are used with Endevco's Model 2775A signal conditioner. The signal conditioner is a charge converter with sensitivity, gain, and test function controls.

1.4 Electronic Components

A block diagram of the analog circuit is in Figure X.1. Due to a positive DC voltage offset, the signal conditioner is AC-coupled to the analog circuit board; the two inputs are then summed into the input amplifier. From the summing amplifier, the signal goes to a true RMS-to-DC converter. To increase the dynamic range, the converter's logarithmic or decibel output option is used. Next, a 10-bit analog-to-digital converter is used as the interface to the computer board. Three-state buffers are needed to move the 10 bits of data onto the computers 8-bit bus. The machining operation signal used by the counter is controlled by the NC machine program and sent through an optical-isolator. The counter output goes to a display on the faceplate and to the computer boards parallel input port. Reset control of the counter is from the computer's output port.

The computer board has a large number of capabilities; the foremost of these is fast floating-point arithmetic that can handle 80-bit numbers. There are 34K bytes of memory available on this board, of which 2K can be nonvolatile memory. To communicate to external devices there are two input ports, two output ports, and a serial port. To help in real-time operation, there are six external interrupts. A block diagram of the microcomputer circuit board is in Figure X.2. As an aid in the understanding of the overall concept of the microcomputer circuit board, the following discussion is divided into four sections, each one containing a number of circuit descriptions. The subjects, in the order they will be described, are microprocessors, control, memory, and I/O.

1.4.1 Microprocessors

Two microprocessors are implemented on this board. An 8088 in the maximum mode is used as the primary Central Processing Unit (CPU), and an 8087 Numeric Data Processor (NDP) is used as a coprocessor. In the maximum mode the 8088 can address 1 megabyte of memory. When an 8087 is used in this manner, the programmer generally does not see it as a separate device; instead, the computational capability of the 8088 appears greatly expanded. The 8087 adds a fast floating point arithmetic capability to the circuit. An increase in speed of 50 to 100, depending on the type of function, can be expected. Data as large as 80 bits, approximate decimal range 0.3 4x10-4932 < x < 1.1x10 4932, are valid values for the 8087 NDP.

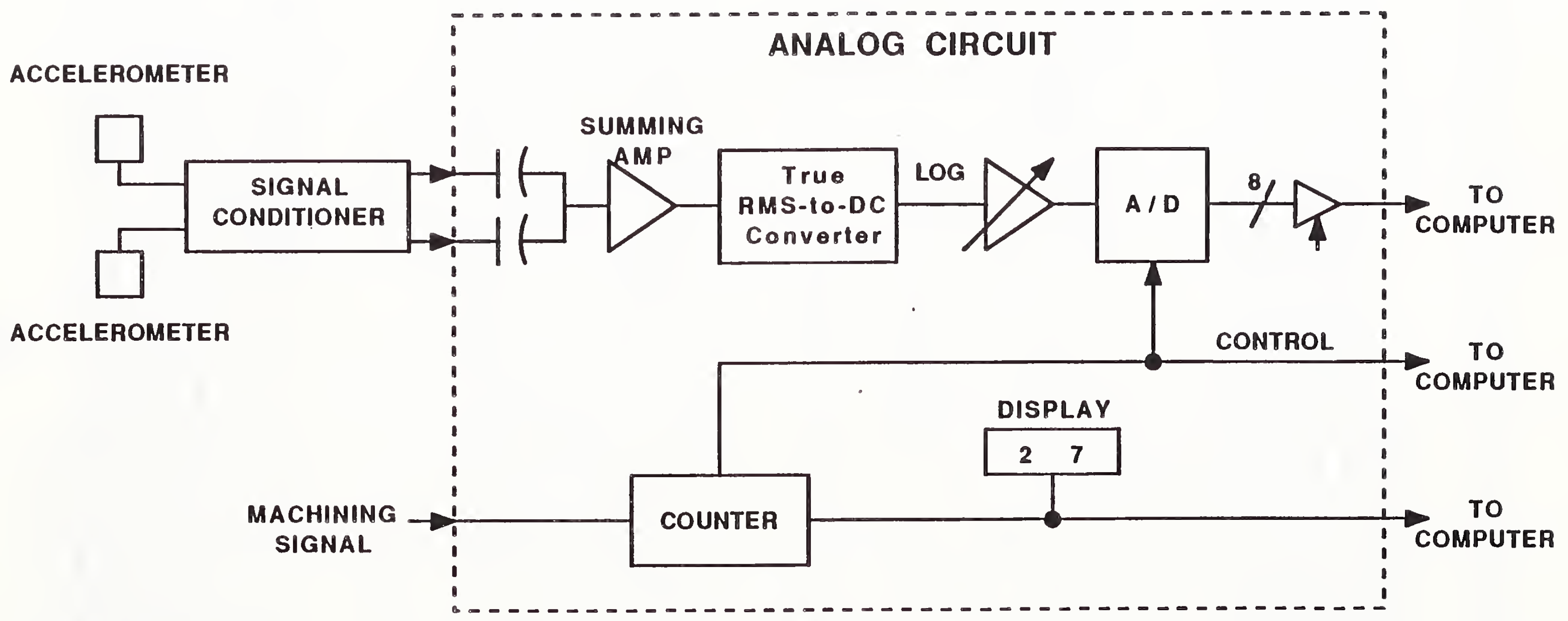


FIGURE X.1. MALFUNCTION AND TOOL CONDITION DETECTOR

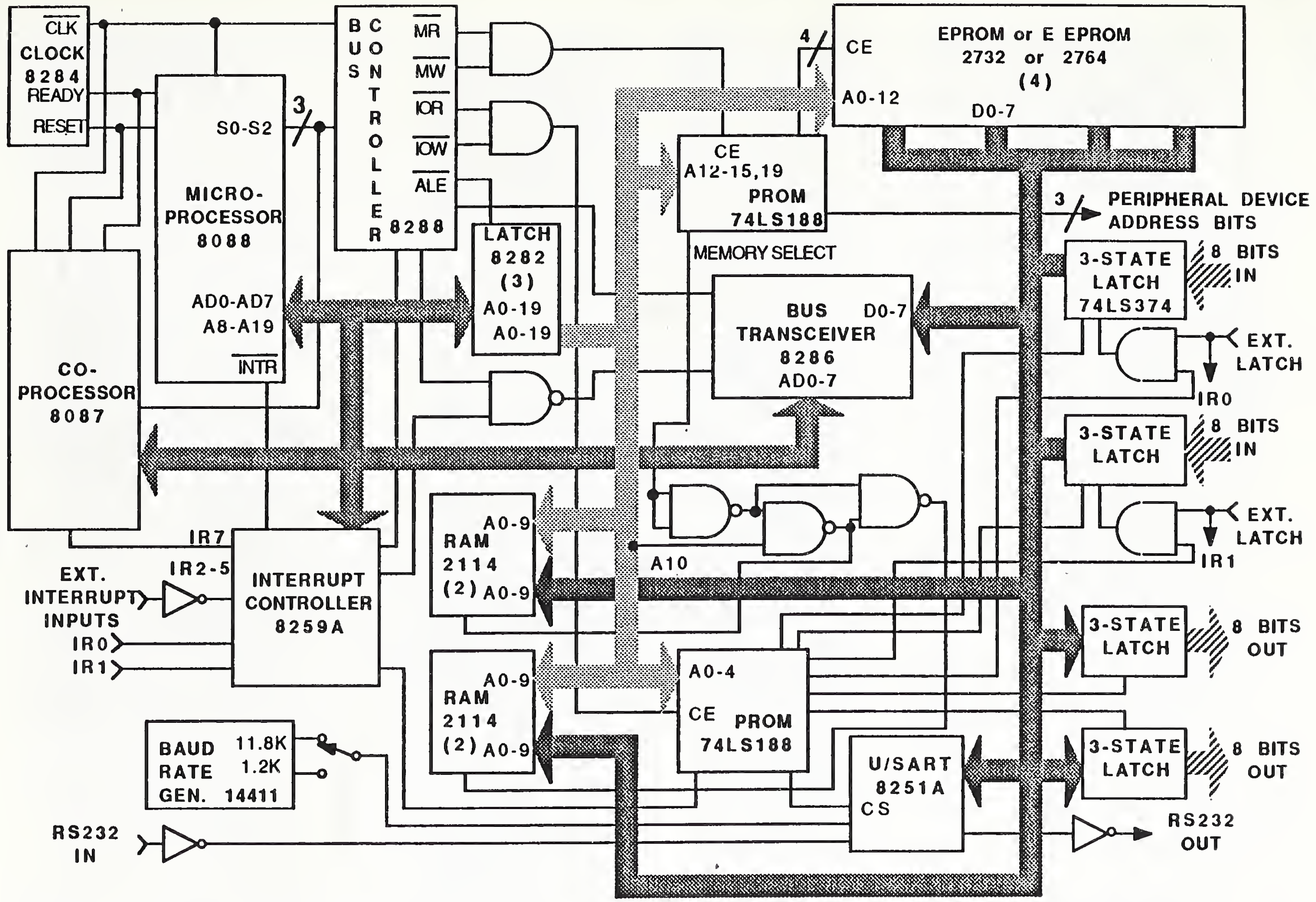


FIGURE X.2. SCHEMATIC DIAGRAM FOR THE NBS-DESIGNED MICROCOMPUTER

1.4.2 Control

The control circuit is composed of several integrated circuits. Two crystal controlled circuits are used to generate the clocks needed by the microprocessors and the serial I/O circuit. The status of the microprocessor is monitored by a bus controller; using this information, it generates the needed memory and I/O control command signals. Three latches use the address latch enable signal (ALE), supplied by the bus controller chip, to create the address bus. Programmable read-only memory chips and an NAND gate decode the address bus, then enable the proper memory or I/O chips. Because the CPU uses a multiplexed bus, a transceiver is needed to isolate the address-data bus from the data bus. An interrupt controller circuit is used to establish interrupt priorities when both an 8087 and interrupts are to be used by the CPU. In this design there are six interrupts available, two of which are also external input commands that latch the input data bus.

1.4.3 Memory

There are three different types of memories that are used, random-access memory (RAM), erasable programmable read-only memory (EPROM), and electronically erasable programmable read-only memory (EEPROM). 2K bytes of RAM memory is available for use. The amount of EPROM memory available is determined by the type used. If 2732's are used, without using an EEPROM, there will be 16K bytes of EPROM, when using 2764's there are 32K bytes of EPROM. The advantage of EEPROM memory is it can be addressed in a similar manner as RAM, but unlike RAM memory it will retain its data when power to the chip is removed. The disadvantages are slow writing speeds and low chip byte densities. A 2816 EEPROM was used because it is almost pin-for-pin compatible with the 2732's and 2764's. If a 2816 EEPROM is used, the total amount of EPROM is reduced by 4K when using 2732's, and 8K when using 2764's. The reason for the EPROM size change is the EEPROM uses one of the sockets normally used by an EPROM chip. The 2816 EEPROM will hold 1K bytes of data. In the latest version of the device, battery backed-up RAM was substituted for the EEPROM. This replaced the EEPROM memory on a chip-for-chip basis and offered the advantage that no special procedures were required to write to this RAM, which in operation is indistinguishable from the other RAM.

1.4.4 Input/Output

There are two types of I/O's used on the CPU board: serial and parallel. The serial input-output is accomplished using a programmable Universal Synchronous/Asynchronous Receiver/Transmitter (USART). This circuit converts 8-bit parallel data to 8-bit serial data for transmission at the same time it also converts received serial data to parallel data. There are two baud (bits per second) rates available, 1.2K or 4.8K. Interfacing to a terminal, using buffers to change the logic levels, is the function of this serial I/O circuit. Four 8-bit parallel ports make up the parallel I/O circuit, two input and two output. The two input ports data can be latched internally or externally, then read by the CPU at its convenience. The two output ports are bit addressable by the CPU.

1.5 Software Components

The software consists of a program, written in PL/M and linked to library I/O routines to form one module. The program consists of two modes of operation, a local mode and a remote mode. The local mode is used for testing and diagnosis as well as the introduction of the tolerance parameters into the memory. It is in this mode that the characteristics of each process are stored. The remote mode operates when signals from the machine tool accelerometers and the machine tool controller are sent to the microprocessor for analysis and malfunction detection.

The essence of malfunction detection consists of comparisons of vibrations, as measured by the accelerometers at any instant in time, with upper and lower limits of vibration for that particular machine operation. These upper and lower limits must be stored and, for convenience, are stored in nonvolatile random-access memory. The stored limits, along with a significance factor, are grouped with each group representing one machine tool program. They are stored in the order of the machine tool operations that will occur during the particular program. The present software provides for up to fourteen different machine tool programs, but this is a current limit due only to memory constraints. The locations of the beginning and final limits for any particular program are stored in yet another table in the nonvolatile RAM memory, along with three timing limits.

In normal operation, a controlling computer sends a character, "s" to "x" or "S" to "Y" to the detector, thereby designating which program the machine tool is to start running. Upon receiving the program designation, the detector returns a "b" to the controlling computer indicating that it is busy. The detector uses the program designation to locate the table of parameters to be used as limits and starts a timing loop while waiting for a signal from the machine tool, indicating that it has started to perform a particular operation. If the timing loop is traversed more than a limit, called the dormant limit, before the signal from the machine tool is obtained, an error signal is sent to the controlling computer.

Upon receiving the signal from the machine tool that it has started the operation, the count of the timing loop traverse is restarted and the loop is again traversed while the accelerometer output is monitored. The highest level of accelerometer output is maintained as well as the last 200 readings. If at any time the accelerometer output exceeds the maximum value associated with this particular step in this particular program, an error message is immediately transmitted to the controlling computer. If the maximum value is not exceeded, but the count on the traversing of the timing loop exceeds a limit called active, an error signal is sent to the controlling computer. If neither the maximum limit is exceeded nor the active timing limit exceeded, upon receipt of a signal from the machine tool that the particular operation has finished, the maximum accelerometer reading obtained is compared with the minimum limit for the corresponding operation in the particular program. If the maximum accelerometer reading is below the minimum limit, an error signal is sent to the controlling computer. When the signal indicating the finish of the particular operation is received from the machine tool, the count of the timing loop traverse is again restarted and the count maintained until receiving the signal of the start of the next step. If at any time the count exceeds a limit called quiescent, an error signal is sent to the controlling computer. These operations continue until all of the steps of the machine tool program have been performed, when an "r" is sent to the controlling computer.

The error signal sent to the controlling computer consists of ten ASCII characters, the first character being an "e". The second character is a digit 1 to 6, indicating the following error designation:

1 The highest accelerometer reading was below the minimum limit established for that particular operation in that particular machine program.

2 The highest accelerometer reading exceeded the maximum limit established for that particular operation in that particular machine program.

3 The allotted time was exceeded.

4 Errors 1 and 3.

5 Errors 2 and 3.

6 The highest accelerometer reading was below the minimum limit established for an operation that would indicate that a part had most likely been dropped.

The added information provided by error 6 as compared with error 1 is provided by the significance factor which is stored along with the minimum and maximum limits. The next three characters are a number, indicating which operation was on at the time of the error. The final five characters consist of a number that is the value of the last accelerometer reading obtained. Whenever the controlling computer does not comprehend the transmission sent by the detector, it can send an "n", which would cause the detector to repeat the previous character. Whenever the detector receives a transmission that it finds illegal or inappropriate for the present circumstance, it issues an "n".

Two variations to the normal operational mode exist. The first variation is used when brand new parameter limits for the current operation are to be obtained automatically. In this mode, the maximum accelerometer readings for any operation in any machine tool program are stored for the first five times that the operation is performed. After the fifth run, the parameter limits are calculated and used for malfunction detection. Until this mode is changed, which requires a shutdown of the detector, each operation causes an update of the corresponding limits. The second variation is used when it is desired to continually update the existing detection limits based on current operating measurements.

Malfunction detection is based on the maximum vibration reading for each turning cut or pass. These maximum vibration readings are defined as values $V_1, \ldots, V_i, \ldots, V_n$. The method of calculating the new limits under the first variation, consists of summing these values and the squares of these values for each cut. Let N be the number of values summed. The running average and running upper and lower limits for any particular cut are then determind as follows:

> Value N is the number of maximum reading values.
> SUM is the sum of the N values.
> MEAN = SUM/N.
> D is the sum for the squares of the last N values of V_i.
> SIGMA = square root [(D-(squares(SUM)/N)/(N-1)].
> Lower limit = MEAN - 3 * SIGMA.
> Upper limit = MEAN + 3 * SIGMA.

With the use of these calculations, the malfunction limits can gradually change as a result of gradual shifts in the machining operation, such as tool wear. Despite the fact that these limits are continually changing, sudden shifts in the vibrations of the machining operation will be interpreted as a malfunction. This is based on the premise that real malfunctions cause sudden shifts in the vibration level.

2. OPERATION OF THE MALFUNCTION DETECTOR

After the powering up of the malfunction detector, the control program waits for a character, which will inform the microprocessor whether to go into local or one of three remote modes. Control-A (binary 1) causes the detector to go into local mode. Characters "U", "#" or "$" cause the detector to go into remote mode. "U" causes the detector to use previously stored limiting parameters (parameters keyed in while in local mode). The character "#" causes the detector to use previously learned limiting parameters and to modify these parameters in accordance with the running mean and running standard deviation method. The character "$" causes the detector to start fresh with no limiting parameters and to then produce these parameters and modify them as described above. Modes of operation can only be changed by restarting the detector.

2.1 Local Mode

In local mode, communication between a dumb terminal and the microprocessor is carried on in English, primarily with the microprocessor asking questions in English and the operator at the terminal responding with "y" or "n". In addition, the microprocessor can request the input of parameter values, in which case an ASCII representation of the number is accepted by the microprocessor. In this mode, everything that the operator types at the terminal is echoed to the screen to facilitate human communication. Remote mode can also be simulated in local mode when needed for diagnosis, but, in general, the operator can accomplish anything desired in the more user friendly local mode.

The following pseudocode displays the operation under local mode. All messages sent by the detector are enclosed in single quotes and responses in double quotes with items within square brackets, "[ ]", being optional. Items between which choices must be made are separated by the vertical bar, "|", while dummy values are shown as "< >" and nonoperable comments will be shown within braces, "{ }". Numbers followed by a colon, ":", are labels.

```
        "Control A"
1:      'YOUR AVERAGING VALUE IS < >'
        'CHANGE AVERAGING VALUE?'
        if "y"
                'IF THE COUNT IS CHANGED, YOU MUST START THE LEARNING PROCESS ALL
                OVER'
                'CHANGE AVERAGING VALUE?'
        if "y"
                'KEY AVERAGING COUNT VALUE'
                "< >"

        'DISPLAY LAST 200 CAPTURES?'
        if "y"
2:              '< > < > ........< >'

        'DISPLAY SUB PROGRAM START & STOPS'
        if "y"
                '< > < > ......< >'
                'CHANGE ANY VALUES?'
        if "y"
3:              'WHICH SUB PROGRAM?
                "< >"
                'START?'
                "< >"
                'END?'
                "< >"
                '
                'CHANGE ANY VALUES?'
                if "y" goto 3
                'DISPLAY SUB PROGRAM STARTS AND ENDS AGAIN?'
                if "y" goto 2

        'GET RMS VALUES?'
```

{ This routine obtains readings as though the detector were operating in remote mode and summarizes the collected data displaying the summary on the terminal.}

```
        if "y"
```

{ The detector now proceeds to collect data until either sixty-four separate machine tool operations have been performed, or until a key is struck on the terminal. The information is then sent to the terminal in groups of ten. }

```
4:      { loop }
5:      'MCODE = < >  COUNT = < >  MINIMUM = < >  MEAN = < >  MAXIMUM = < >'
        .....................................................................
        .....................................................................

        'HIT A C/R TO RESUME'
        " Carriage return " goto 5
```

{ MCODE is the number of the machine tool procedure and COUNT is the number of readings that were taken to produce the summary. }

```
        'REPEAT PRINT-OUT?'
        If "y" goto 4

6:      'DISPLAY AND/OR CHANGE LIMITS?'
        if "y"
                'LEARNED LIMITS?'
                "y or n"
                'WHICH PROCESS?'
                "< >"
                'LOW LIMIT IS  < >    HIGH LIMIT IS  < >[Involved in dropped part
                detection]'
                'SET NEW LIMITS?'
                if "y"
                'LOWER LIMIT?'
                "< >"
                'UPPER LIMIT?'
                "< >"
                'CHANGE DROPPED PART ACTIVITY?'
```

{ This refers to whether or not a too low signal at this program step will suggest the existence of a dropped part. }

```
        if "y"
                'MAKE ACTIVE?
                "y" | "n"
        goto 6

        'DISPLAY TIMES?'
        if "y"
                'DORMANT = < >  QUIESCENT = < >  ACTIVE = < >'
        'TIME SETTING?'
        if "y"
                'BY TIMING?'
        if "y"
                'HIT A KEY TO START AND THE SPACE BAR TO STOP'
```

```
 "any key then space"
       '< > DONE'
       if "n"
              'BY KEYING IN?'
       if "y"
              'MINUTES?'
              "< >"
              'SECONDS?'
              "< >"
              '< >'
if "y" { for timing or keying in}
       'STORE IN DORMANT?'
       "y or n"
       'STORE IN QUIESCENT?'
       "y or n"
       'STORE IN ACTIVE?'
       "y or n"
'DISPLAY TIMES?'
if "y"
       'DORMANT - < >  QUIESCENT - < >  ACTIVE - < >'

'TAKE READINGS'
if "y"
       'HIT A KEY TO START AND THE SPACE BAR TO STOP  f DOUBLES SPEED OF
           OPERATION'
```

{ When any key is hit, the detector will start taking accelerometer readings at 5 second intervals except if the key was an f, in which case the interval will be 2 and 1/2 seconds and then prints each value to the terminal. The operation is not under any machine tool control and is used mainly for testing or diagnosis of problems. Keying the space bar halts the process. }

```
       'CHECK/CHANGE ERROR PRINT TYPE?'
       if "y"
              'Errors are in ASCII'
              'Change to BINARY?'
              "y | n"
                      |
              'Errors are in BINARY'
              'Change to ASCII?'
              "y | n"
```

{ One of the above two options will occur but not both. }

```
       goto 1
```

2.2 Remote Mode

In remote mode, all communication with the microprocessor consists of one ASCII character, not echoed by the microprocessor, with the exception of the error signal, which consists of a string of ten ASCII characters. The remote mode follows the following communications protocol:

'b'	Busy
'c'	Clear
'e'	Error
'i'	Inquiry
'n'	Not understood
'r'	Ready
'z'	Understood
's'	Start program #1
't'	Start program #2
'u'	Start program #3
'v'	Start program #4
'w'	Start program #5
'x'	Start program #6
'y'	Start program #7
'S'	Start program #8
'T'	Start program #9
'U'	Start program #10
'V'	Start program #11
'W'	Start program #12
'X'	Start program #13
'Y'	Start program #14

The following pseudocode displays the operation under remote mode:

```
        "U" | "#" | "$"
1:      'r'
        "s" | "t" | "u" | "v" | "w" | "x" | "y" | "S" | "T" | "U" | "V" | "W" |
    "X" | "Y"
        'b'
```

{ The device performs malfunction detection of the relevant program (1 to 14) under one of the three modes until the machine program is finished. }

```
      { if no malfunction detection }
            goto 1
      { if malfunction }
           'e < > < > < > < > < > < > < > < > < >'
           goto 1
```

This constitutes the entire communication during remote mode. If the first character sent to the device is "U", then the malfunction detection is performed using the relevant predetermined parameters. If the first character sent is a "#", then the malfunction detection is performed, using the relevant previously learned parameters, which will continue to be updated. If the first character sent to the device is a "$", the device will immediately try to learn new relevant limiting parameters with no malfunction detection being performed during the first five operations of any program step. Subsequent to the fifth operation, the device will perform in a mode identical to the mode started by "#".

3. SUGGESTIONS FOR FUTURE DEVELOPMENT

A possible future development would entail using the malfunction detector to indicate tool wear along with the malfunction detection. The malfunction detection is based on a sudden change in the accelerometer signal level. It is highly likely that there is a slight continuous change in the accelerometer signal level for a particular process due to tool wear. If this were the case, statistics on this tool wear could be obtained using the malfunction detector. Parameters could be established based on these statistics, thus permitting timely warnings relevant to tool wear.

XI. COLLET CHANGER

Tooling and tool management represent one of the key issues when designing and operating a flexible machine center. To allow untended operation for a turning center, a system for changing collets has been implemented on the turning center at the AMRF. This is the first known use of a robotic system to change a standard collet on a turning center. Collets are needed for precision machining to tolerances of 100 microinches.

1. DESIGN OF THE COLLET CHANGER

1.1 Overview

In order for the robot to change a collet, a specifically designed computer-controlled collet-changing device is installed on the NC turning center. The controller designed for this device is highly sensory interactive. Its basic function is to loosen or tighten the collet and properly engage or disengage the clutch assembly.

1.2 Control Architecture

The controller block diagram is given in Figure XI.1. At the heart of the controller is an 8-bit 8751 microcontroller. A software program in the microcontroller handles the job of generating the pulse width modulation (PWM) timing cycles necessary to drive the motor, monitor torque, activate the clutch mechanism, and read sensor input.

Two types of input are available on the collet changer, an RS232 serial link and manually activated switches located on the lathe. The primary input is the RS232 link to the workstation controller (WSC). The WSC can then command the collet changer to change collets. In return, the collet changer provides status and error information.

The controller status may take on one of the three values: ready, busy, and error. While the status is ready, the controller will accept commands from the RS232 link. During the busy state only one command is accepted. This is the reinitialize command. When the controller receives the initialize command, it aborts its current operation and resets all internal variables.

If an error occurs during operation, the collet changer will broadcast an error status and stop whatever operation it was performing. When this status is acknowledged, the controller ceases to transmit status information and outputs the most recent error code. At this point, the controller will remain inactive until it is commanded to either re-initialize or to continue with its current operation.

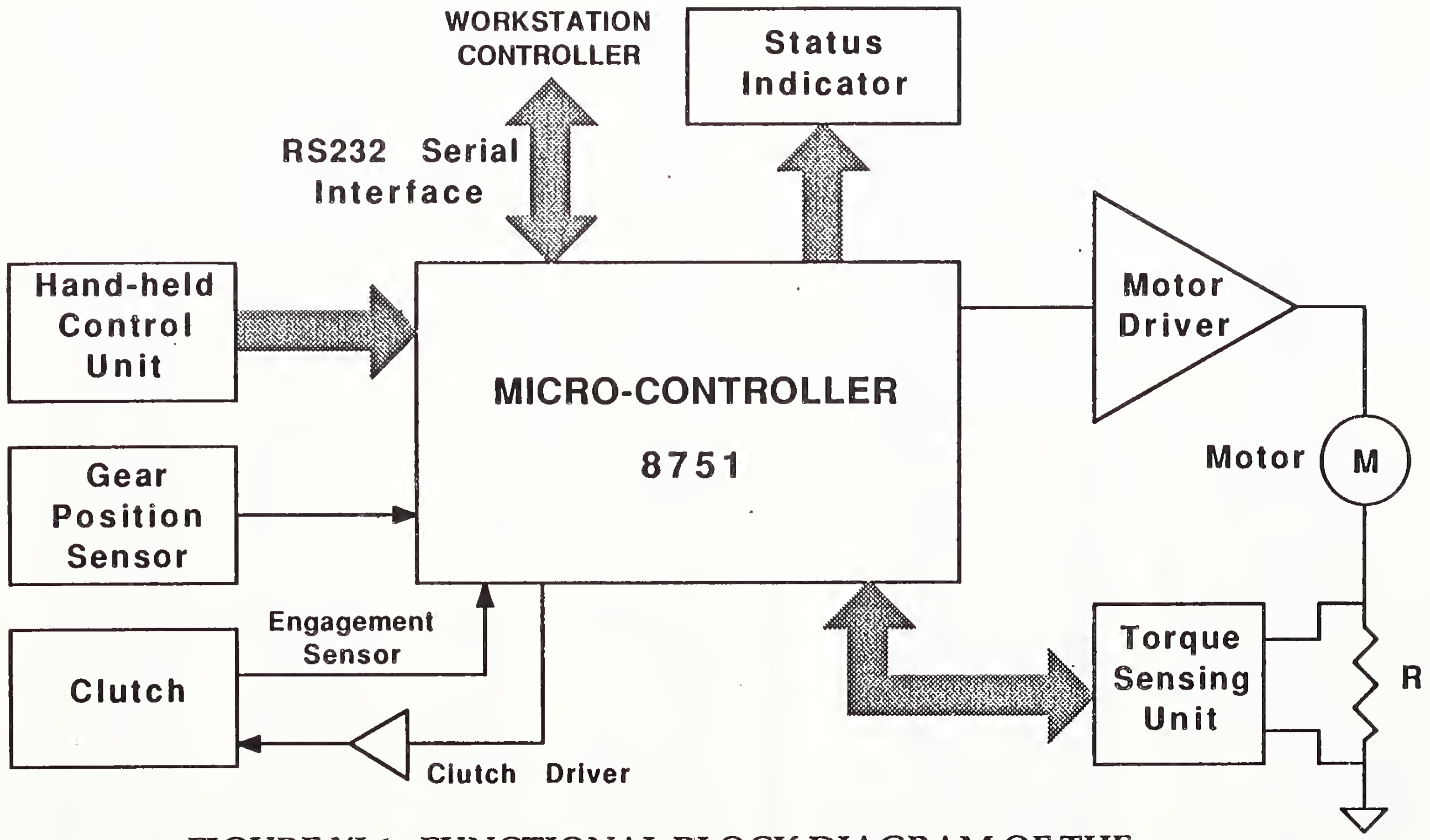


FIGURE XI.1. FUNCTIONAL BLOCK DIAGRAM OF THE COLLET CHANGER CONTROLLER

1.3 Mechanical Components

The collet changer is described below using the following diagrams. Figure XI.2 is an isometric view of the spindle, collet changer, and robot arm grasping a collet. Figure XI.3 is a cross-section of the collet changer mechanism.

The collet changer consists of a clutch which is actuated by an air piston (3). The clutch mechanism is then driven by a D.C. motor (11) through a gear train (4). Induction sensors (7) are used to count and align the clutch mechanism to the spindle. A further discussion of the mechanical components will be given in the section on the operation of the device.

1.4 Electronic Components

The heart of the collet changer controller is the 8-bit 8751 microcontroller which has 128 bytes of RAM for data memory and 4K bytes of EPROM for program memory. The microcontroller sends out PWM pulses to drive the servomotor and actuates a relay to change motor direction. The motor's torque is monitored by the microcontroller sampling the current passing through the motor. An A/D converter is used to sample the motor current and to convert the analog value to 8-bit data. This data is read by the microcontroller.

1.5 Software Components

The software routines consist of an algorithm to tighten or loosen any type of collet. It is a "smart" system which can correct itself under certain difficult conditions.

There are five errors which may occur. These errors are communication, time out, binding, alignment failure, and clutch failure. A communication error occurs if the collet changer receives an erroneous command. A time out error occurs if the changer fails to receive feedback after a specified time. A binding error occurs if the drawbar becomes difficult to turn at a point in the operation where this should not occur. This may be due to either mechanical binding or cross threading of the collet. An alignment error will occur if the changer is unable to align the clutch properly. Finally, a clutch error will be generated if the clutch fails to return to its proper location after being activated. One can get a feel for what the software is doing by reading the section on the operation of the device.

2. OPERATION OF THE DEVICE

The automated collet changer utilizes a typical collet changing operation. In order to loosen a collet (9, Figure XI.2), a pivoting fork (2) is actuated by an air piston (3, Figure XI.2). This will disengage the locking pins (13, Figure XI.3) to free the spindle from the drawbar (1, Figure XI.3) and also engage the pinion (6) to the drive gear (5) of the clutch mechanism. The pinion is driven through a gear train (4) attached to a DC servomotor (11). A

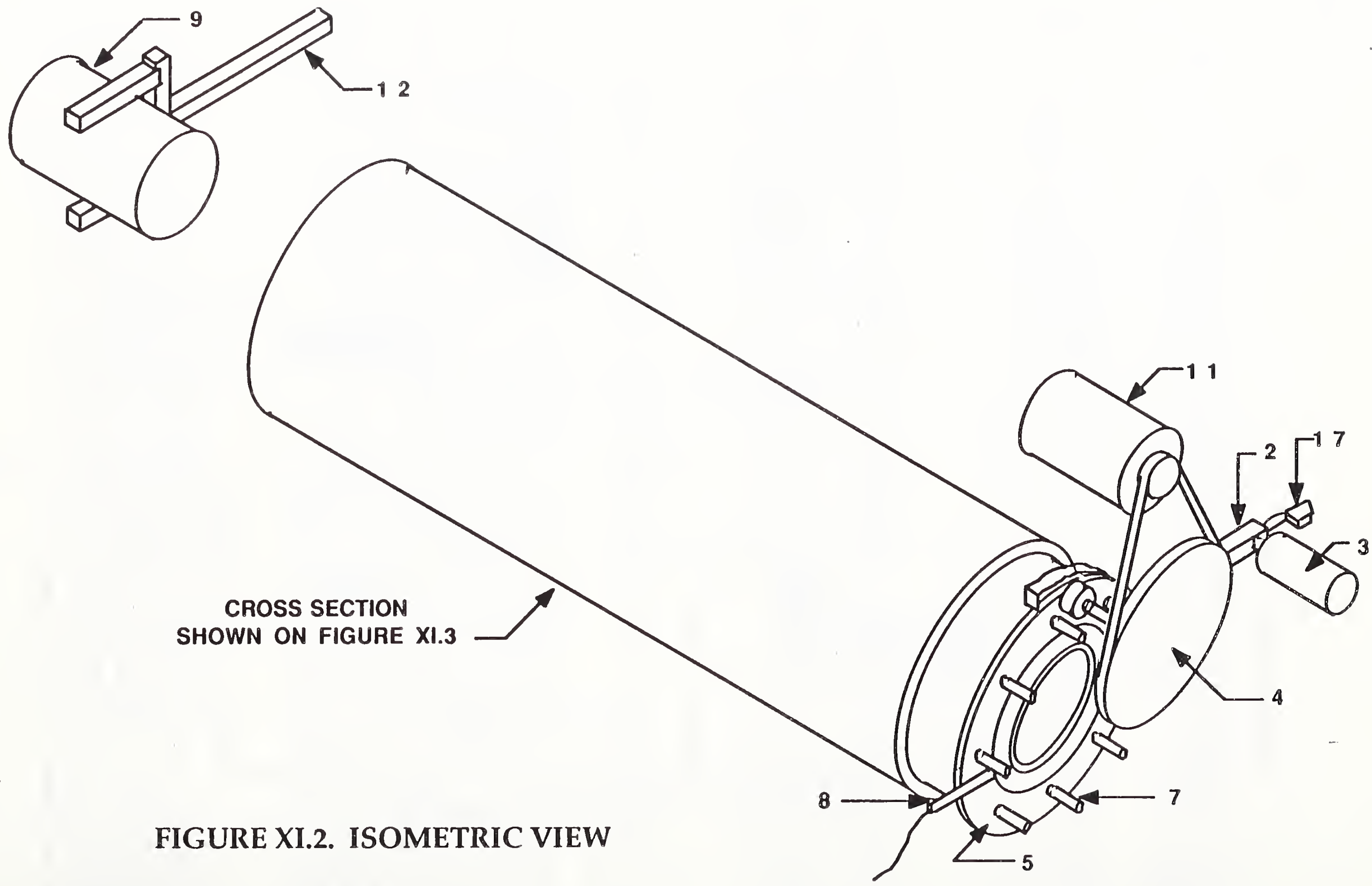


FIGURE XI.2. ISOMETRIC VIEW

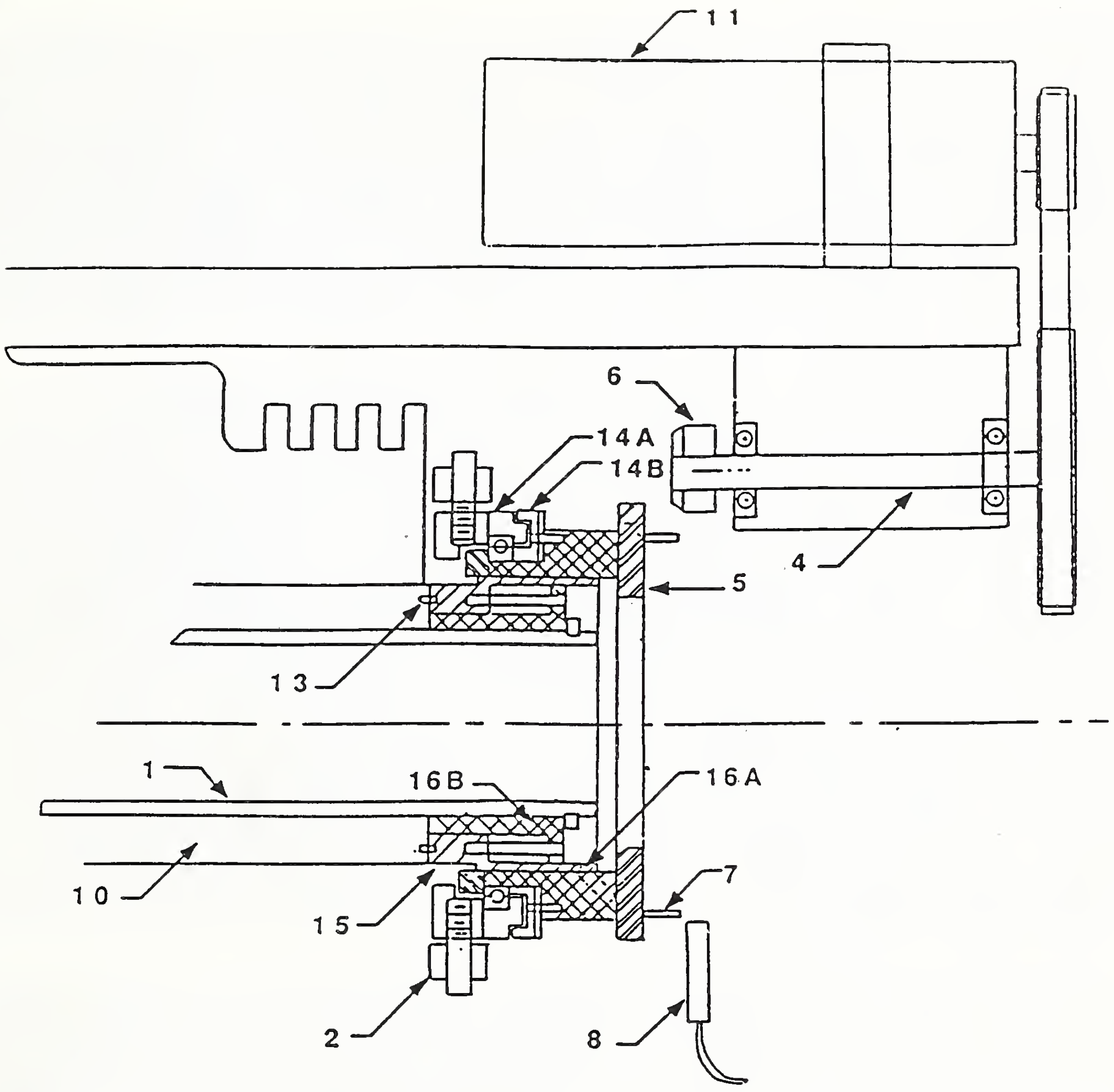


FIGURE XI.3. CROSS SECTION OF CLUTCH MECHANISM

nonrotating linear move of the clutch is made possible by four equally spaced guide pins (15, Figure XI.3), and a close tolerance sliding sleeve (16A and 16B, Figure XI.3). While the clutch is being engaged, the motor controller pulses the servometer in order to properly mesh the gears. Eight equally spaced steel locating pins (7) are placed on a diameter around the driven gear. An induction sensor (8) is used to count and read the position of each of these steel pins. The induction sensor is not affected by cutting oil and coolant that have been known to migrate up the spindle and disperse around the headstock. Likewise, a labyrinth bearing seal (14A and 14B) was designed to safeguard against any contamination of the bearings.

With the motor engaged, the induction sensor is used to count the appropriate number of turns of the drawbar to loosen the collet. When the collet is disengaged from the drawbar, the robot (12, Figure XI.2), can remove the collet and insert a new one. The shaft of the collet aligns the collet with the spindle throat so the drawbar will not crossthread. While pushing on the collet with the compliant device (attached to a programmable tailstock, not shown) the drawbar is turned by the DC servomotor. The controller, using PWM drive at low speed, counts 32 pulses of the induction sensor to rotate the drawbar four full turns. After the four turns, the collet will be initially threaded to the drawbar. By monitoring the motor current, using an analog-to-digital converter, the motor controller can sense the proper torque on the drawbar in order to seat the collet in the spindle properly. Once the collet is snugged up to the spindle, it has to be backed off one-half of a turn. This will give the collet proper clearance to load a workpiece and actuate the drawbar. Having the collet too tight can cause damage by exceeding its elastic limit. Likewise, a loose collet would not be able to hold a workpiece securely against the large forces placed on it during turning operations. To back the collet off one-half of a turn, the motor is reversed and the motor controller counts back three counts.

The above scenario is similar to the way a machinist would install a collet for safe use. Since the spindle locking pins (13, Figure XI.3), are lined up with the locating pins, the clutch is now ready to be actuated to lock the drawbar to the spindle. If in any case, the spindle locking pins are not engaged, a limit switch (17, Figure XI.2) will signal the controller to search again. If the spindle cannot be locked after three trials, an error message is sent to the workstation controller to allow for the manual intervention. Such a scheme makes this system smart enough to prevent turning operations, if the spindle is not locked. The collet is now in position to be loaded with a workpiece. (Robot loading of a workpiece in a collet is made possible by a micromanipulator developed at NBS). After the workpiece has been inserted by a robot gripper, the drawbar is actuated to clamp the workpiece in the collet. Actuation of the drawbar moves the clutch assembly 0.1 to 0.25 inch closer to the pinion; however, this amount of travel is not enough to engage the clutch. Spindle speeds of up to 4000 rpm can be expected, so one has to be confident that the pinion will never be engaged during turning operations. The clutch assembly can compensate for the drawbar travel by the compression of air in the air piston.

This design permits the changing of any type of collet, either dead length or regular. There was minimum modification to the turning station. It is compatible with bar feeding operations, or any other operations where the spindle must remain clear. It is in this manner that a computer controlled adjustable stop could be installed on a turning center at NBS. This system has been found to be a great asset in manual machine operations, making it quicker and easier for a machinist to change a collet. Communications to the controller can be made via the workstation controller, or through manual overrides located on the lathe.

3. SUGGESTIONS FOR FUTURE DEVELOPMENT

Since the collet changer is a prototype, some components can be improved to get better performance and a longer service life. Using a spline bushing instead of the sleeve bearing would provide a better way of transmitting torque in the mechanism. Chrome plating could also be used on moving parts to reduce wear and corrosion. Status lights or information provided to the CRT on the CNC lathe would help for manual operation of the collet changer. As it stands now, that information is only given to the workstation controller.

XII. TURNTABLE

As a buffer storage device for the TWS, the turntable gives the workstation robot unimpeded access to items that are used in workstation operations. Up to sixty-six tools, collets, and gripper fingers total can be stored on holding fixtures on the turntable. These storage locations are arranged in three concentric circles from which a needed item is elevated by one of three pneumatically actuated shafts. The elevation of the part by the shaft allows the robot gripper an unobstructed path to the item, as well as repeatable pick-up and drop-off points.

1. TURNTABLE DESIGN

1.1 Overview

The turntable consists of a 36-inch diameter platter that is horizontally mounted on a frame approximately 3-feet high. The platter, with room for up to sixty-six holding fixtures, is belt-driven by a servo-motor. The platter is rotated to the correct position upon receipt of a command from the workstation controller. Pneumatically activated shafts are used to lift the holding fixtures up off the platter allowing free access to the robot gripper. The system is microprocessor-controlled and uses an absolute-position encoder for feedback of the platter's position. In the following discussion, the word "turntable" will generally refer to the whole system - mechanical and electromechanical - whereas "platter" will refer to the actual 36-inch diameter turning platter.

Implementing the buffer storage device as a servo-controlled turntable allows for a compact, simple design which performs two functions. The first function is that of a buffer storage unit for the turning center and for the robot. In the course of machining a part in the turning center, different collets may be used. Ordinarily, these collets would be retrieved, changed, and stored manually. However, when using a robot to change collets, there must be a storage area where an unneeded collet can be dropped off and a required one picked up. Other items which would be put in buffer storage are extra tooling for the turning center and the various gripper fingers used by the robot for picking up different parts. Each of these types of implements would have its own type of holder, but all holders would fit on the platter in the same way - that is, any holder could be installed on any of the sixty-six positions of the platter.

The second function of the turntable is to provide unobstructed, repeatable pick-up and drop-off points for the robot. The shafts used by the turntable to lift the holding fixtures above the platter provide height repeatability as well as a clear path to and from the fixture for the robot. The standardized holders (for each type of tool or implement) provide position and orientation repeatability for the tooling. Repeatability is especially important for the pick-up operation. If the orientation or pick-up point of a collet, for example, is slightly off, then it may not be possible to insert it into the spindle of the turning center.

In addition to the microprocessor and support circuitry, the system consists of an absolute encoder for position feedback and a DC servo motor driven by a commercially obtained motor controller. The controller software is stored in EPROM and is written in PLM-51, a high-level language written for the 8051 family of Intel microprocessors. The software consists of different routines used for servo-control, error checking, and communication with the workstation controller.

1.2 Control Architecture

The turntable is operated using a basic servo-control algorithm. The principal requirement that this algorithm fulfills is that of accurate rotation of the platter to the required locations for the raising and lowering of the desired holding fixture. While rotating the platter, its current position is checked periodically and compared with the desired destination. When these two positions coincide, the turntable controller waits for a command from the workstation controller to raise (or lower) the appropriate shaft, in order to raise (or lower) the required tool holder.

In order to satisfy the requirement of accurate platter positioning, the combination of DC servo motor, motor controller, and absolute encoder was chosen. This combination allows the platter to reach the needed destination as quickly as possible and it has the advantage of the encoder's constant absolute position output.

Control of the turntable platter is accomplished by using two nested feedback loops. The inner feedback loop consists of the motor, tachometer, and a commercially obtained motor controller. The microprocessor-controlled outer loop consists of an absolute-position encoder, a D/A (digital-to-analog) converter, and the commercial motor controller.

The inner loop controls the speed and direction of the motor based on the command and tachometer signals to the motor controller. The motor controller (which was commercially obtained) outputs a high-voltage PWM (pulse-width modulated) signal to the motor. On the basis of feedback from the tachometer (which is attached to the motor) and the command signal input, the motor controller varies the speed and direction of the motor, accordingly.

In the outer loop, the absolute encoder is used to position the platter, and is connected directly to the platter shaft. This eliminates the effects of backlash between the shaft and the drive belt, because the encoder is intended to indicate the actual position of the platter, not the position of the motor shaft. The resolution of the encoder is 2048 counts per revolution, which translates to a resolution of about 0.055 inches at the outer edge of the 36-inch diameter platter. During a move from one position to another, the microprocessor reads the encoder and constantly updates the platter position information.

In controlling the turntable, the microprocessor sends values to the D/A converter which are converted (at the output of the D/A) to voltage levels of the proper polarity. This voltage output is the command signal, which is sent to the motor controller to control the speed and direction of the motor driving the platter. As the platter is rotated near the desired position (destination), the microprocessor reduces the command signal, causing the platter to decelerate. When the destination is reached, the command signal is reduced to zero, and the platter stops. If an overshoot occurs, the microprocessor directs the D/A to produce a small command signal of the opposite polarity to return to the destination position.

Once the platter is in position, it can be "locked" at that position by servo control or the motor can be de-energized, leaving the platter in position, but free to move. In either case, the turntable controller microprocessor waits for the command from the workstation controller to raise the proper shaft. While the shaft is being raised (or lowered) the position of the platter is checked and corrected, if necessary. The shaft remains in the raised position until the workstation controller commands the turntable controller to lower it.

1.3 Mechanical Components

The turntable is described below using the following diagrams. Figure XII.1 is an isometric view of the machine frame and working parts. Figure XII.2 is an illustration of a typical application.

Referring to Figure XII.1, the mechanism consists of a 36-inch diameter turntable (1) which pivots on two centering bearings (2A and 2B). The machine frame (7) is made from 6-inch, steel channel welded as shown. This design was chosen to achieve maximum compactness, rigidity, and ease of service. Three adjusting bolts (7A) are provided, one on each leg, to easily level the turntable relative to the shop floor. Sixty-six stations are available with a capacity of 200 pounds for tooling and fixtures. The table is driven by a totally enclosed DC servo motor (3A) and motor controller (3B). All electrical and pneumatic components fit into a service area inside the frame. Position feedback is accomplished through an 11-bit absolute encoder (4). To avoid any backlash problems the encoder is coupled directly to the driven shaft (6). This gives a theoretical resolution of 0.003 in/in radius of the turntable. The DC motor and controller gives the advantage of having an infinite selection of stop locations throughout the axis range.

Tools are placed on three different radii on top of the turntable. There are three linear guides underneath the table. These consist of linear bearings (5A) and nonrotating shafts (18A) which raise the tool off the turntable. The turntable rotates to a desired tool where it will then be in position for actuation of the appropriate linear guide. The linear guides are actuated by pneumatic pistons (8), placed parallel to the shaft of the linear guides. The

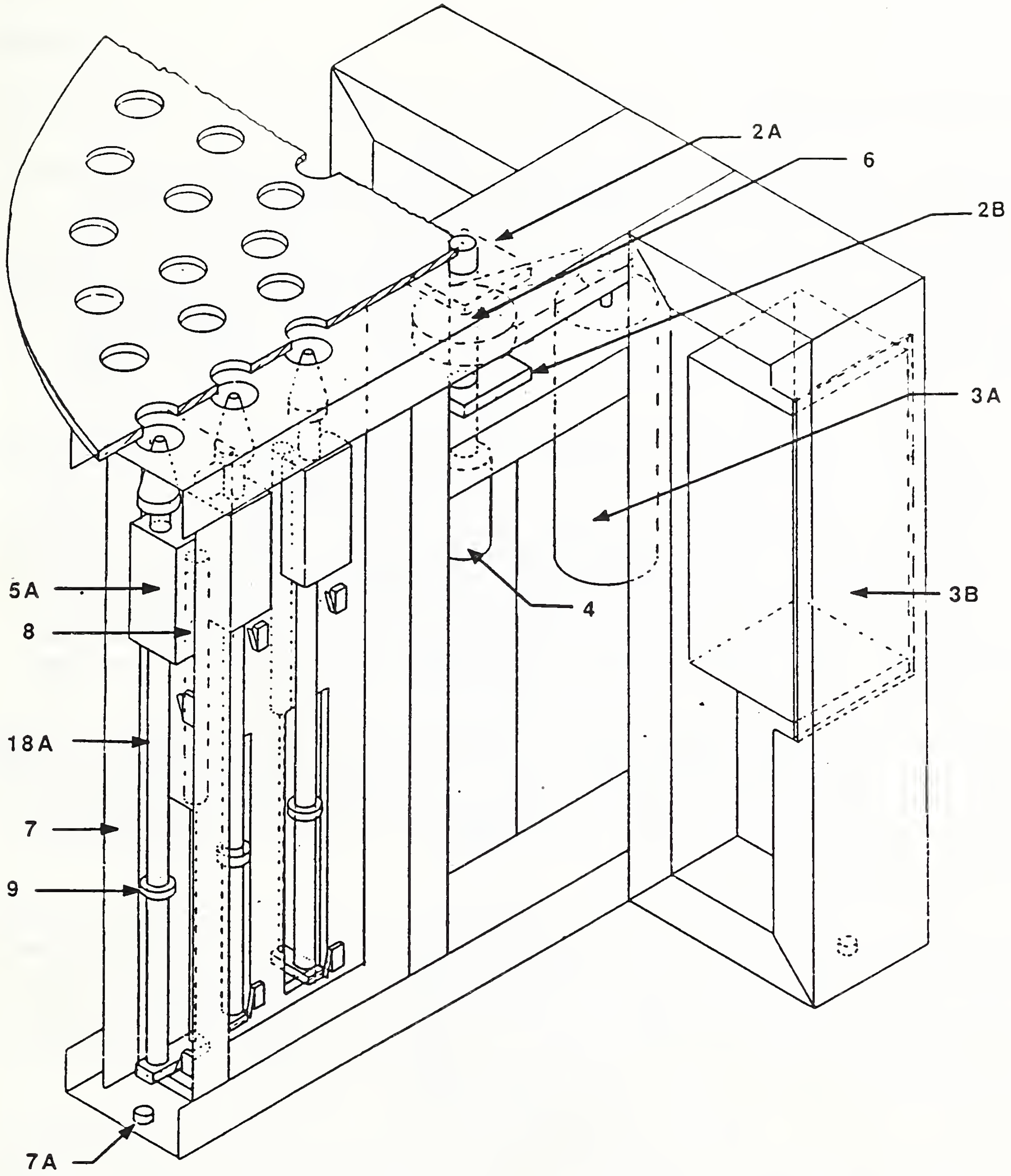


FIGURE XII.1. ISOMETRIC VIEW OF THE TURNTABLE MACHINE FRAME

stroke on each shaft can be easily adjusted by a squeeze collar (9), to be used as a mechanical stop.

Referring to Figure XII.2, when the linear guide is going up, the male machine tape (10A) engages the female taper (10B) of the tooling fixture, which is mounted on top of the turntable. A keyway cut into the male taper is also engaged by a pin in the female taper. In this manner, the proper orientation of a tool would automatically be accomplished as it is being raised and finally the proper position will be set when the linear guide hits the mechanical stop (squeeze collar) to end its travel. Speed control of the linear guides is adjusted by standard muffler-speed controls which are attached to a common manifold for easy access.

To illustrate a typical tool changing operation, again refer to Figure XII.2. The robot (12) needs only one teach point to access any tool on a radius. When replacing and retrieving a collet (14), for instance, the gripper will go to that teach point with the old tool in its fingers. The empty tool fixture is indexed to the correct position to be picked up by the linear guide. (To save time, this move can be done anytime before the gripper comes over.) The linear guide is actuated and the tool is guided into the fixture. Tapered fixtures (15) are used to help center the placement of the tool in the fixture. A passive compliance system (16) is also used between the tapered base and the tool fixture. This compliance will allow proper transfer of tools even when small forces are applied to the fixture due to misalignment.

After the linear guide reaches the end of its travel, a limit switch (17) is actuated, signalling to the controller that the guide is up. In this way the machine is made "smart enough" to prevent the turntable from rotating when a guide is in the raised position. This signal is also used as feedback to the robot controller to tell it that the tool fixture is in the correct position for the transfer of a tool.

Now, the robot gripper releases the old tool and the linear guide is retracted. Tapered pins (19A) will line up with locating holes (19B) to hold the fixture on the turntable. Limit switches are used to indicate when a linear guide is in the down position, allowing the table to index to its next position. The turntable rotates to the new tool, the linear guide is actuated, the gripper fingers are closed, and the linear guide retracts. The robot now has the new tool and the old tool is catalogued on the turntable for future use.

Using the turntable and linear guide to index and pick up tools the following can be accomplished: (1) a linear move by the robot is eliminated; (2) excellent repeatability is attained through the use of mechanical stops and the mating of a machine taper; (3) a compact design is feasible (large number of tools per area); (4) there is ample clearance for the robot gripper to work during tool transfer, despite high tool storage density; and (5) the use of robot memory is kept at a minimum since only three teach points must be stored, one for each radius.

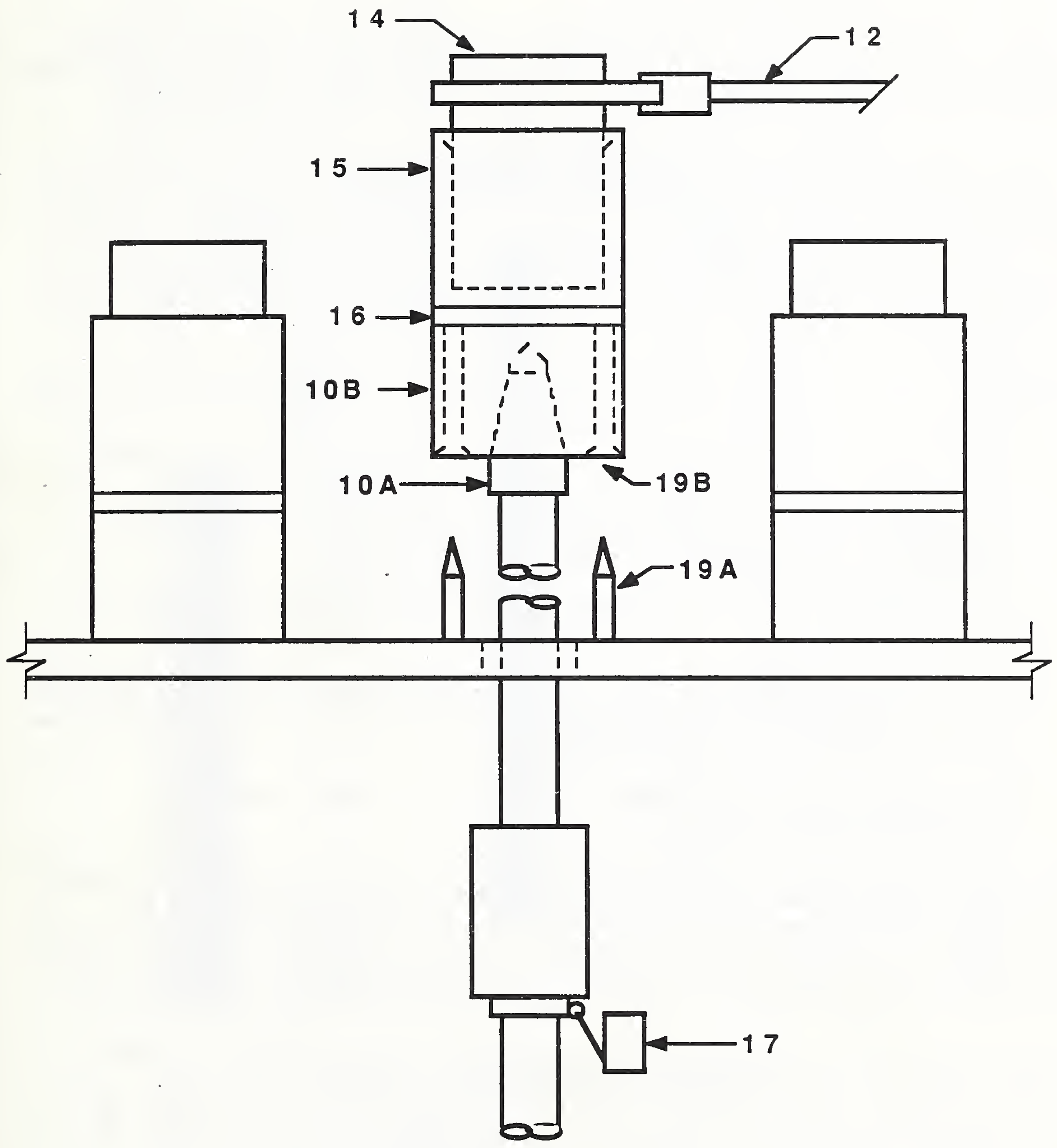


FIGURE XII.2. A TYPICAL APPLICATION OF THE TURNTABLE

1.4 Electronic Components

The electronic components of the turntable controller (illustrated in Figure XII.3) consist of the microprocessor and its peripherals, an absolute-position encoder, the motor/tachometer unit, and the commercially obtained motor controller. There are also various switches, relays and solenoid valves.

An Intel 8751 microprocessor was chosen for use in the controller. It is the EPROM (Erasable Programmable Read Only Memory) version of the 8051 family of microprocessors which is supported by PLM 51, a high-level language, and has development tools for testing, developing, and debugging the design. The 8751 has an 8-bit data line, bi-directional ports, timer/counters, and external interrupt lines. There are 128 bytes of volatile memory and 4K (4096) bytes of program memory located on-chip.

Some considerations which went into the choice of a microprocessor for this job are :

- the availability of development tools to debug the initial prototype;

- whether or not the microprocessor is supported by a high-level language (this is not necessary, but it does make the writing, debugging and documentation of the software easier);

- whether or not the I/O (Input/Output) ports are easy to control and manipulate;

- the kind of communication that is intended between the microprocessor and the workstation controller and whether or not the microprocessor can accomplish this kind of communication;

- whether a sufficient number of I/O lines, counter/timers, and interrupts are present.

The I/O lines of the microprocessor have byte or bit control capability, which makes them easy to use for this design. A serial communication port is included in the microprocessor, which makes serial (RS232) communication with the workstation controller an easy task. Although no external interrupt lines were needed, two timers were necessary - one to set the serial communication baud rate, and the other to set the cycle time for the servo routine.

Since there are not enough I/O lines to handle the large number of input and control signals, a port expander chip is used in conjunction with the microprocessor. This chip, the Intel 8155, contains two full I/O ports (16 bits total) plus part of another. The extra ports are used mainly for the 11-bit encoder, and the chip is controlled by the microprocessor.

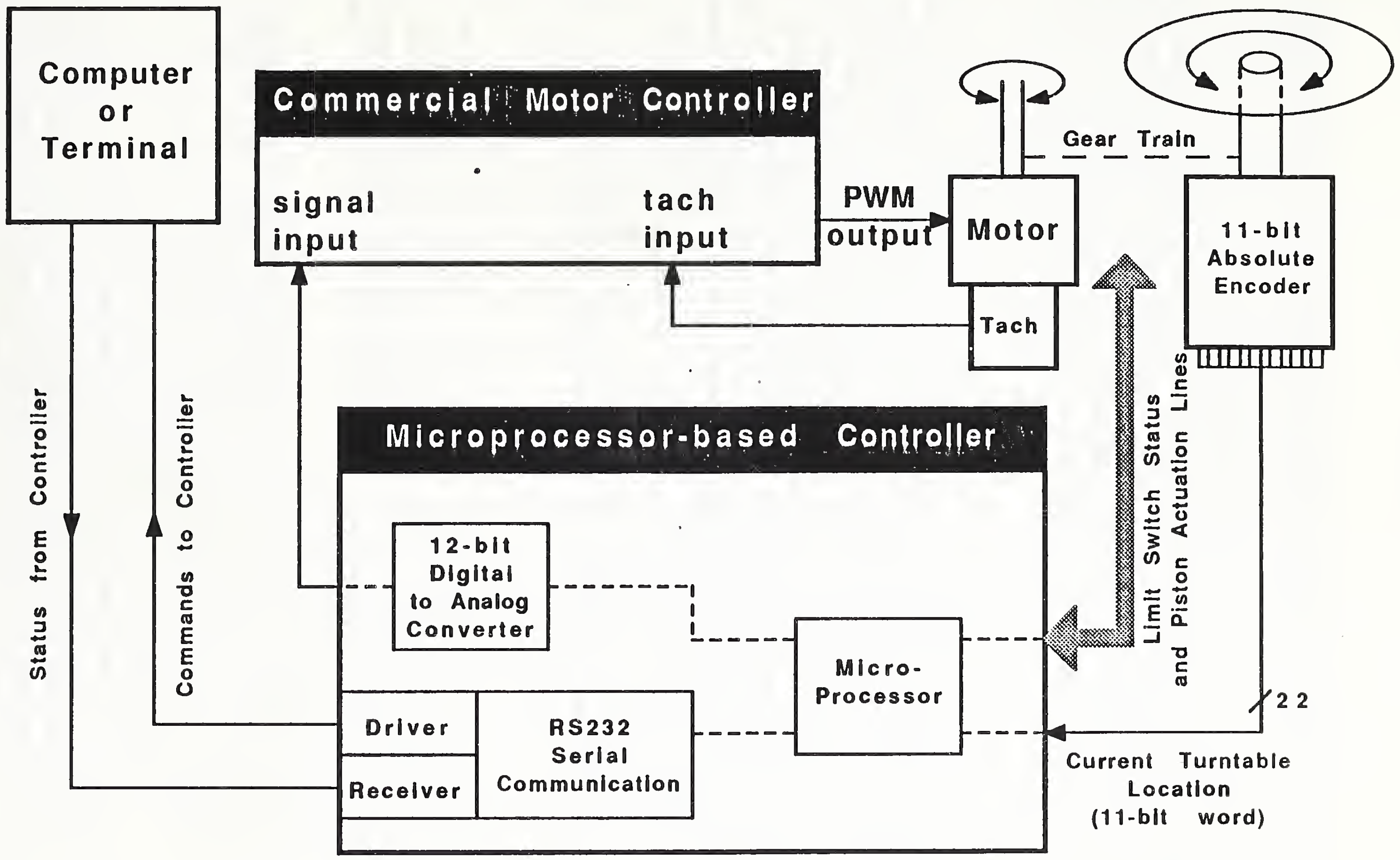

Computer or Terminal
Commercial Motor Controller
signal input
tach input
PWM output
Motor
Tach
Gear Train
11-bit Absolute Encoder
Limit Switch Status and Piston Actuation Lines
22
Current Turntable Location (11-bit word)
Microprocessor-based Controller
12-bit Digital to Analog Converter
Micro-Processor
Driver
Receiver
RS232 Serial Communication
Status from Controller
Commands to Controller

FIGURE XII.3. FUNCTION BLOCK DIAGRAM OF THE TURNTABLE CONTROLLER

The 12-bit D/A converter, the AD567 (Analog Devices), converts the digital command from the microprocessor to a voltage command signal which is sent to the motor controller. A low offset operational amplifier (AD544, Analog Devices), is used to buffer the output of the D/A converter.

The absolute-position encoder is an 11-bit absolute encoder (M25 series, BEI, Inc.). The required resolution of the encoder was decided based on the worst-case acceptable positioning resolution of the platter. It was determined that the position error at the edge of the platter (worst case) could not exceed 0.1 inch, so for a 36-inch diameter platter, an 11-bit (or 2048 count) encoder was chosen. This works out to 0.055 inch per count at the edge of the platter.

The absolute encoder was chosen for two reasons. First, the position information is always available to the microprocessor - it doesn't get reset or wiped out when the power goes off, or when the system is reset. This means the microprocessor will never lose track of where the platter is, and will not have to find a "home" position whenever the power is cycled. The incremental encoder, by comparison, does not provide absolute information. It would be necessary for the microprocessor to add and subtract pulses to determine the platter position. The position information would be lost when the power is cycled. The second reason is that the software appeared simpler for the absolute encoder than for the incremental encoder. Using absolute numbers to compare the current position with the destination seemed easier than keeping track of the number of pulses and the phase relationship of the two signal lines (three, including the index pulse) of an incremental encoder.

The motor is a permanent magnet DC servo motor which is supplied with a tachometer. It maintains a continuous (stall) torque of 960 ounce-in, and a peak torque of 4500 ounce-in.

The motor controller is a PWM servo controller which is capable of +/- 15 amps continuous output current to the motor. It accepts feedback from the motor tachometer and a control voltage from an external source; in this case the microprocessor control circuit. The motor controller also has an externally controlled motor shutdown line. This line is controlled by the microprocessor and is independent of the command voltage. The controller also has adjustments for gain, current-limiting, and balance.

The weight of the platter and its contents dictate the size considerations of the motor, and thus the motor controller. Fully loaded with tool holders and tools, the platter would weigh about 200 pounds, and a large motor was employed. A smaller one may have worked as well, however, especially if the platter speed were kept low and adequate ramping were used.

The three pneumatically operated pistons are operated using air solenoids (+12 VDC). Piston positions are detected by microswitches, a pair to each piston. One of the pair detects when the piston is fully raised, the other detects when the piston is fully lowered. The six microswitch signals are monitored by the microprocessor at all times. Also used for piston operation, as well as for the motor shutdown function, are reed relays (located on the circuit board with the microprocessor) and larger, enclosed relays located at the turntable fixture.

The turntable controller is implemented using an 8751 8-bit microprocessor, which contains 4K bytes of erasable memory for program storage. Because of the large number of input and output signals, a port expander chip is used. This increases the number of ports the microprocessor has for I/O use. The 11-bit encoder information, the input to the D/A converter, and the shaft status information are multiplexed onto the 8-bit data bus of the 8751. The control lines (i.e., the shaft control lines and the motor shutdown lines) originate from the microprocessor. The output of the D/A converter controls the direction and speed of the platter motor through the commercial motor controller.

When power is applied to the turntable controller (the motor controller has its own power switch), the microprocessor ensures that all pistons are down and the motor is deactivated. This condition is designed to be the normal state. Whether or not the microprocessor circuitry is powered, the motor drive is deactivated, as are the piston solenoids. This prevents the pistons from inadvertently raising, or the platter turning, if there is an interruption or glitch in the power. It also acts as a safeguard against unforeseen software malfunctions. There is much less chance of the platter "running away" or the pistons raising if the normal condition for these operations is deactivation.

The microprocessor begins monitoring the position of the platter and the status of the piston micro-switches. Because there were not enough I/O lines to use one for each of the six status switches, the six lines were coded into three lines. A PROM (Programmable Read Only Memory) decoder chip, the 74lS188, and an OR-gate were used to represent the pistons' five states as a 3-bit binary code. The five states are:

1. piston no. 1 up,
2. piston no. 2 up,
3. piston no. 3 up,
4. all pistons down,
5. anything else is an error, i.e., more than one piston up at a time.

These three lines, as well as the eleven lines from the absolute encoder, are connected to the 8155 port expansion chip, which is connected to the microprocessor's 8-bit bidirectional data bus. In this way, the microprocessor is constantly monitoring the shaft status and encoder information.

The D/A converter is also connected to the microprocessor's data bus. The D/A, in its turn, is constantly being updated by the microprocessor . Between platter moves, and at power up, the microprocessor sends a value to the D/A which corresponds to a voltage of zero. This voltage (at the output of the D/A) is the command signal, which is sent to the motor controller. A voltage of zero prevents the motor from turning. Note that this is a redundant safety measure, because the motor shutdown is also in effect at this time, as previously mentioned.

The microprocessor communicates with the workstation controller over a serial (RS232) link. When the turntable controller receives a command from the workstation controller to move the platter to a certain location, the appropriate direction (CW or CCW) is calculated, the motor driver is activated and the microprocessor sends a value to the D/A converter corresponding to a voltage with the correct magnitude and polarity to move the platter. As the platter moves, approaching its destination, the encoder is read every 30 ms. The platter speed is reduced in proportion to the distance remaining. When the destination is reached, the D/A output (command signal) to the motor controller is, again, zero volts, which stops the motor. The microprocessor then disables the motor drive (the "shutdown" condition).

Normally the workstation controller would then send the command to raise a shaft. The microprocessor will first confirm that the platter is still in the proper position, then it will cause the proper piston to be raised. The piston status switches are monitored to determine when the shaft is completely raised. The sequence is repeated when the workstation controller commands the shaft to be lowered.

To operate (raise or lower) a piston, the microprocessor activates (or deactivates) a small reed relay. This reed relay, which is located on the circuit board with the microprocessor, in turn activates a larger relay located at the turntable. This relay actuates the air solenoid which causes the shaft to be raised or lowered. This configuration buffers the circuit board from the (noisier) turntable environment.

A relay is also used in the motor shutdown circuit. When the relay, which is located at the turntable, is deactivated (its normal state), it grounds the shutdown control line of the motor controller, disabling the motor. To activate the motor drive, the relay is actuated, allowing the SHUTDOWN control line to remain open.

1.5 Software Components

Software for the turntable is written in PLM-51, a high-level language for the Intel 8051 family of microprocessors. The program is linked with the PLM-51 library, compiled and stored in the on-chip program memory of the 8751, occupying approximately 3800 bytes of code.

The software is implemented using a real-time control structure. The position and status of the platter are checked periodically (every 30 ms). A routine is used to slow the platter as the destination is approached, and hold it in position (or disable the motor) when reached. Error conditions are constantly checked for and a routine is used for implementation of serial communication.

There are two interrupt procedures. Both are called by internally generated interrupts. One interrupt is generated when a data flow is detected into or out of the microprocessor's serial port register. The other interrupt is generated when a timer (within the microprocessor) overflows. This timer is pre-loaded with a value which causes it to overflow every 30 ms. When this overflow interrupt occurs, it causes program execution to exit the main loop and begin the timer interrupt procedure. This procedure directs the microprocessor to take periodic readings of the encoder and shaft status lines, calculate the distance from the current platter position to the destination, and update the D/A converter.

Flags and variables are also updated. At the end of the interrupt procedure, program execution jumps back to the main loop and continues where it left off, but with updated information.

The process begins at power-up (or reset) with a setup procedure which disables the motor, makes sure all shafts are down and initializes all registers and control variables. This includes loading the internal baud rate timer and the 30 ms timer (above) with the proper values, setting the interrupt priorities in case they both occur at the same time, configuring the serial communication register, and finally, starting the timer and enabling the interrupts.

When the setup routine is completed, the program begins executing the repetitious main loop, jumping out temporarily because of a timer overflow interrupt (every 30 ms) or a serial communication interrupt.

Whenever the workstation controller sends a command to the machine tool controller, the characters of the command are stored, character by character as they are received, in a buffer area of memory. Each time a character (i.e., a letter or a control character) is received by the microprocessor's serial communication register, an interrupt ("serial interrupt") is generated. When this happens, execution temporarily jumps from the main loop to the serial interrupt procedure. Here, they are placed as they are received, into a buffer area until the complete command is assembled. Since the characters are being transmitted at a rate of 1200 baud, there is plenty of time for the microprocessor to return execution to the main loop until the next character is received and generates another interrupt. The incoming command is complete when a carriage-return character has been received. The interrupt procedure places this command in a command buffer and sets a "command pending" flag, to be read in the main loop. After execution jumps back to the main loop and this flag is found to be set, a procedure is called to determine which flag it is and how to respond to it. If it is an unrecognized command, an error message is generated, and sent to the workstation controller. This command-handling procedure also determines what numeric parameter, if any, was sent with the command.

Each command has its own subprocedure. The current command (including numeric parameter, if present) is matched with the correct sub-procedure, or command procedure. These command procedures first call another procedure to send an acknowledgement character back to the workstation controller. This acknowledges that the command has been received. Another character will be sent (later) upon completion of the action called for by the workstation controller. The procedure then checks for out of limit numeric parameters, and sends an error message, if necessary. If the command is a request for information ("What's the current position?") the procedure will send out the correct response and jump back to the main loop. If the command is a call for an action ("Move the platter to location 54.") the procedure will set flags and update variables to be read later in the main loop, then initiate the action, and jump back to the main loop.

The main commands sent by the workstation controller to the turntable are:

1. move the platter to another position,
2. raise the shaft,
3. lower the shaft.

As mentioned previously, the platter has sixty-six storage locations, arranged in three concentric circles. Each storage location is defined by two pieces of information: the encoder value when that location is lined up over the proper shaft, and the number of the proper shaft. This information was determined (by hand) for each of the sixty-six locations, and stored in a look-up table. (Note that if the encoder is removed or slips relative to the platter, this look-up table will have to be altered.)

Suppose, for instance, the turntable is commanded to move the platter to position 37 (the workstation controller keeps track of the contents of each location). The program looks up the thirty-seventh item on the list of sixty-six platter locations and finds the proper shaft to activate (later) and the encoder reading for that location. This encoder reading now becomes the DESTINATION variable, and the current encoder reading is compared to DESTINATION to determine how far and in which direction to turn the platter. The command procedure for this command would give the proper values to the DESTINATION and SHAFT_NUMBER variables, set a flag to indicate that the platter is on its way to a new location, activate the motor drive and return back to the main loop. Every 30 milliseconds the main loop is interrupted by the routine that reads the encoder, calculates the distance remaining, and updates the D/A converter. The command signal sent to the D/A converter decreases as the distance to go decreases, until the platter stops. If the platter overshoots, or creeps past the destination, the polarity of the command signal is reversed, and the platter returns.

When the turntable receives the "raise shaft" command from the workstation controller, it looks at the current value of the SHAFT_NUMBER variable, and activates that shaft. When the command to lower the shaft is received, that shaft is lowered. As before, when each command is received, program execution jumps to the proper command procedure, where an acknowledgement is sent and the action is initialized. In the case of the shaft command, flags are set, to be looked at in the main loop, that indicate that a shaft is on its way up (or down). The status switches for that shaft will be monitored to confirm that it's all the way up (or down) before sending an "action completed" message to the workstation controller. Also, the position of the platter is checked whenever a command to raise or lower the shaft is received. If the platter is not at the correct position (it may have been moved) it is adjusted before the shaft is allowed to move up or down.

2. OPERATION OF THE TURNTABLE

2.1 Local Operation

2.1.1 Setup Requirements

An RS232 terminal is used for local control and operation of the turntable. Incoming data is received on line 2, and data is sent to the terminal over line 3. Line 7 is used as the signal ground, as is standard. The terminal should be set to full duplex mode, no parity checking, and with a baud rate of 1200. In local mode, the characters are echoed back to the terminal, and a carriage return/line feed response is included.

2.1.2 Commands and Responses

The turntable commands are listed below. The ENTER (or carriage return) key must be pressed to complete each command, because the software looks for it before setting the "command pending" flag in the program loop. If the command includes numeric parameters, they are indicated within < >. The standard response to any command from the workstation is a "b" (for "busy") upon receipt of the command, and an "r" (for "ready") after completion of the command. In local mode these character responses are each followed by a line feed (LF) and a carriage return (CR).

In the following discussion, the quote marks shown are not included with the command, nor are the blank spaces. For example, "b LF CR" is actually sent bLFCR , and gh CR is entered ghCR.

ec <0 or 1> CR   Echo

This command sets or resets the ECHO flag. When the ECHO flag is set (1), which it would be for local operation, the command sent to the programmable stop is echoed back to the terminal screen, with a LF (line feed) character. When reset (0), there is no echo or LF. The response to this command is a "b CR LF" upon receipt and a "r CR LF" after completion.

gl <1 through 66> CR   Go to Location < >

This command moves the platter to the location specified by the number. The proper shaft for this location is automatically determined. This command is to be used with the "xp" and "lp" commands for raising and lowering the shafts. The response to this command is a "b CR LF" upon receipt and a "r CR LF" after completion.

xp CR   Extend Shaft

This command is used only in conjunction with the "gl< >" command. It causes the shaft which was defined by the latest use of the "gl< >" command to be raised. The response to this command is a "b CR LF" upon receipt and a "r CR LF" after completion.

Lower Shaft

This command is usecconjunction with the "gl< >" command. It causes the shaft which was y the latest use of the "gl< >" command to be lowered. The respors command is a "b CR LF" upon receipt and a "r CR LF" after completior

ı or 1> CR   Shutdown the motor < >

If a 1 is entered, und sets a flag which de-activates the output stage of the motor driver,ıg the motor. If a zero is entered the motor is enabled. The respoɒis command is a "b CR LF" upon receipt and a "r CR LF" after completior

R   Display current Position and Destination

This command displa;rrent encoder reading and the current value for the variable DESTINATIOſurrent encoder reading is prefixed by a "p" and is a number from 0 to ie current destination is prefixed by a "d" and is also a number from (. There is no response to this command.

) or 1><0 or 1>   Slide <slide#><up or down>

This is a manual corr raising or lowering a particular slide. For example, "s11 CR" wiꝫ shaft one, and "s20 CR" will lower shaft two. There is no responses command.

<0 through 2047>   Move to location < >

This is a manual corr moving the platter to a specific encoder position, as opposed to one oi defined locations. This command would be used to help define a locatie response to this command is a "b CR LF" upon receipt and a "r CR :er completion.

There are four possi:or messages that can be sent back to the workstation controller. The mesıave the format "e <number> LF CR" , where <number> is the error type. the quotes and spaces are not included.

One error condition ınrecognizable command. This usually occurs when the command is misspelleother occurs when an illegal numerical parameter is used; for example, wnumber greater than sixty-six is used with the "gl" (Go to Location) com A third condition is when the platter is commanded to move with a shaft raised, and a fourth is when a raise shaft command is given when the plis still in motion.

2.2 Remote Operation

2.2.1 Setup Requirements

Remote operation entails communication with, and control by, the workstation controller. The setup requirements are similar to those for local operation, and the same RS232 link is used. Data to the workstation controller is transmitted on line 3, data is received on line 2 and line 7 is signal ground. The difference is that there is no echo back to the workstation controller and the LF character is not included in the responses.

2.2.2 Commands and Responses

The commands are similar in remote mode and local mode. Again, the CR character must follow the command, and the spaces and quotes are not included. The responses, "b" and "r", are the same also, except that the LF character is not included.

ec <0 or 1> CR   Echo

This command sets or resets the ECHO flag. When the ECHO flag is set (1), which it would be for local operation, the command sent to the programmable stop is echoed back to the terminal screen, with a LF (line feed) character. When reset (0), there is no echo or LF. This is the normal power up state and the workstation controller does not normally need to use this command. The response to this command is a "b CR" upon receipt and a "r CR" after completion.

gl <0 through 2047> CR   Go to Location < >

This command moves the platter to the location specified by the number. The proper shaft for this location is automatically determined. This command is to be used with the "xp" and "lp" commands for raising and lowering the shafts. The response to this command is a "b CR" upon receipt and a "r CR" after completion.

xp CR   Extend Shaft

This command is used only in conjunction with the "gl< >" command. It causes the shaft which was defined by the latest use of the "gl< >" command to be raised. The response to this command is a "b CR" upon receipt and a "r CR" after completion.

lp CR   Lower Shaft

This command is used only in conjunction with the "gl< >" command. It causes the shaft which was defined by the latest use of the "gl< >" command to be lowered. The response to this command is a "b CR" upon receipt and a "r CR" after completion.

sd <0 or 1> CR   Shutdown the motor

If a 1 is entered, this command sets a flag which de-activates the output stage of the motor driver, disabling the motor. If a zero is entered, the motor is enabled. The response to this command is a "b CR" upon receipt and a "r CR" after completion.

st CR   Start

This command is used as a software reset. It re-initializes all the variables and registers, and disables the motor drive. The response to this command is a "r CR".

The same error messages are used in remote mode as are used in local mode. The messages have the format "e <number> CR" , where <number> is the error type. Again, the quotes and spaces are not included. See the error description in the local operation section, above, for a list of errors.

3. SUGGESTIONS FOR FUTURE DEVELOPMENT

One problem the turntable has is that the platter "hunts" when serving (or holding) a position. This hunting appears as a rapid shaking of the platter once it is in position - a back and forth movement of about 0.1 inch at the edge of the platter. The problem was dealt with by disabling the motor drive after the platter reached its position, but this solution doesn't afford "holding" the platter in position. Instead, the position is checked and adjusted if necessary before a shaft is raised or lowered. Another solution would have been to mechanically brake the platter after the motor was deactivated.

The hunting problem may have occurred because of noise on the command signal (D/A output) line, insufficient encoder resolution, or a combination of both. Regarding the encoder, an incremental encoder would be seriously considered for a future design. It would be less expensive, which means the same or greater resolution could be obtained for lower cost. If an encoder with a resolution of 4096 counts was used, the hunting would be considerably reduced. Another possibility would be to use a resolver. The circuitry would be more complex than for a digital encoder) but the servo performance would be better. Regarding noise on the command signal line, this prototype design could be improved in the area of grounding, shielding, etc. Better results may be obtained by relocating the D/A converter from the microprocessor circuit board to the turntable itself, near the motor controller. Low-level noise on the digital lines between the microprocessor and the D/A converter would be less of a problem than the same noise on the analog output line from the D/A converter to the motor controller.

APPENDIX A

COMMUNICATION PROTOCOL BETWEEN THE TWS CONTROLLER AND THE DEVICES WITHIN THE TURNING WORKSTATION

A.1 MACHINE TOOL CONTROLLER

In local mode, the high-level machine tool controller communicates with the workstation controller through a serial I/O port. In this mode the database for the NC part programs and tool offset data tables is the local database residing in the workstation controller system. The command and status information between the workstation controller and the high-level machine tool controller is passed from one to the other as single ASCII characters. The following list is a sample group of commands from the workstation controller:

g: loosen collet
h: tighten collet
f: send w axis to home position
k: send x and z axes to home position
l: run the NC part program to make the part

After receiving each command from the workstation controller, the high-level machine tool controller compares it with the set of acceptable commands stored in the memory. If the command is acceptable, it starts executing accordingly and sends a "Busy" status back to the workstation controller. When the task is completed, it sends a "Ready" status to the workstation controller and waits for a new command.

In the remote operation mode, the communications between the high-level machine tool controller and the workstation controller are through the AMRF network. The AMRF database is used in this mode instead of the local database. These communication protocols are similar to those employed in the other stations of the AMRF. Different command and status mailboxes, which are allocated in the common memory, are used in these transactions. Based on the sequence numbers, these mailboxes are transferred by the network server. In this mode, instead of single characters, complete texts are used for each command. Some examples are given in the following:

COLLET LOOSEN	: loosen the collet
COLLET TIGHTEN	: tighten the collet
W HOME	: send w axis to home position
XZ HOME	: send x and z axes to home position
MAKE PART	: run the NC part program to make the part

In this mode, upon power up the operator has to answer some additional questions such as whether he wants to select an NC program list from the database, or upload/download an NC program to/from the database, or reset the mailbox sequence numbers to zero. After the operator goes through this self-explanatory question-answer period, the real operation of the high-level machine tool controller does not differ from the local operation mode. During the operation, an "R" from the keyboard will reset the controller, and a "Q" will show the status information from the current operation. During this inquiry, it is possible to change the status of some of the devices listed on the screen to prevent the controller from getting caught up in an indefinite status request mode while the controller on the other end can not respond due to a malfunction situation.

A.2 ROBOT CONTROLLER

Remote operation entails communication with, and control by, the workstation controller. The RCI commands listed below, used in remote operation, are a subset of the commands available for use in local mode. The CR character must follow the command, and the spaces and quotes are not included. The responses - "b" and "r" - correspond to BUSY and READY,respectively. For the two tray station controller commands, however, the responses are "bt" and "rt", respectively. This is to distinguish the tray station status from the robot status.

cr CR Control Reset

This command activates the Control Reset switch on the robot controller. Control Reset clears the robot controller of the current robot program, and is activated prior to a new program call. The response to this command is a "b CR" upon receipt and a "r CR" after completion.

cs CR Cycle Start

This command activates the cycle start switch of the robot controller. After a robot program is called, cycle start is used to begin execution. It is also used to resume execution during the program if a Motion Hold has been activated. The response to this command is a "b CR" upon receipt and a "r CR" after completion.

mh CR Motion Hold

This command activates the Motion Hold switch of the robot controller. Motion Hold is a temporary halt of robot motion and program execution. Both are resumed with the activation of Cycle Start. The response to this command is a "b CR" upon receipt and a "r CR" after completion.

rt <1 through 999> CR Robot Program Call

When this command is received, a series of actions is performed which result in a robot program being called and its execution begun. A robot controller "end of program" signal is monitored by the RCI. When this signal is detected, robot motion is inhibited. The response to this command is a "b CR" upon receipt and a "r CR" after completion.

st CR Start

This command is a software reset and is used to initialize all registers and inhibit robot movement. The response to this command is a "b CR" upon receipt and a "r CR" after completion.

td CR Tray Deactivate

This command frees the tray from control of the turning center, and allows access to it by the robot cart. The command is typically sent after the turning center has completed a tray full of parts. This command is not specific to the tray number, though it can be made so. As is, it deactivates all trays. The response to this command is a "bt CR" upon receipt and a "rt CR" after completion.

tr <1 or 2> CR Tray Request

This command activates the Request line for Tray 1 or Tray 2. This is done prior to starting a batch of parts in the turning center. The robot cart is thus prevented from changing or moving the tray, as the robot will be picking up blanks and dropping off finished parts to it. The response to this command is a "bt CR" upon receipt and a "rt CR" after completion.

A.3 GRIPPER CONTROLLER

The commands and their respective formats are:

ONCR	open gripper #n, where n is an ASCII character 1 or 2, CR means carriage return.
CNCR	close gripper #n.
PN XXCR	position gripper finger #n to an opening of xx millimeters, where xx are ASCII characters.
SNCR	request status of gripper #n.
ELCR	put gripper controller serial link in echo mode.
EOCR	put gripper controller serial link in non-echo mode.
RNCR	software reset gripper #n.

The responses to these commands include ready and busy signals when the controller is ready to receive a command or is busy executing one. An error condition response informs the host which jaw the error has occurred on and what type of error has occurred.

The responses and their respective formats are:

RNCR	gripper #n is in the ready mode, CR means carriage return.
BNCR	gripper #n is in the busy mode.
EN XCR	gripper #n is in the error mode, where "x" is the error code in ASCII characters:

X=0, illegal command.
X=1, part dropped from gripper.
X=2, bad encoder counts.
X=3, gripper is opened wider than commanded.
X=4, tried to open, but won't move.
X=5, bad limit switch.
X=6, gripper is stuck.
X=7, tried to close, but won't move.

A.4 MALFUNCTION DETECTOR

All communication with the TWS controller consists of one ASCII character not echoed by the malfunction detector's microprocessor, with the exception of the error signal, which consists of a string of ten ASCII characters. The following communication protocol is used:

'b'	Busy
'c'	Clear
'e'	Error
'i'	Inquiry
'n'	Not understood
'r'	Ready
'z'	Understood
's'	Start program #1
't'	Start program #2
'u'	Start program #3
'v'	Start program #4
'w'	Start program #5
'x'	Start program #6
'y'	Start program #7
'S'	Start program #8
'T'	Start program #9
'U'	Start program #10
'V'	Start program #11
'W'	Start program #12
'X'	Start program #13
'Y'	Start program #14

The following pseudocode displays the operation under remote mode:

```
        "U" | "#" | "$"
1:      'r'
        "s" | "t" | "u" | "v" | "w" | "x" | "y" | "S" | "T" | "U" | "V" | "W" |
     "X" | "Y"
        'b'
```

(The device performs malfunction detection of the relevant program (1 to 14) under one of the three modes until the machine program is finished.)

```
        ( if no malfunction detection )
              goto 1
        ( if malfunction )
             'e < > < > < > < > < > < > < > < > < >'
             goto 1
```

This constitutes the entire communication during remote mode. If the first character sent to the device is "U", the malfunction detection is performed using the relevant predetermined parameters. If the first character sent is a "#", the malfunction detection is performed, using the relevant previously learned parameters, which continue to be updated. If the first character sent to the device is a "$", the device will immediately try to learn new relevant limiting parameters with no malfunction detection being performed during the first five operations of any program step. Subsequent to the fifth operation, the device will perform in a mode identical to the mode started by "#".

A.5 MICROMANIPULATOR

When communicating with the TWS controller, all instructions to the micromanipulator controller consist of one ASCII character, which is not echoed by the micromanipulator controller. The micromanipulator controller sends status characters to the TWS controller such as "r" or "b" which indicate ready or busy, respectively. When the micromanipulator controller receives an illegal or inappropriate command, it responds with the character "n" and awaits the resubmission of the command. Whenever a problem occurs, such as the inability of the micromanipulator controller to perform the task commanded, it issues an "e" to the serial port. When it receives an "i" at the serial port, it issues a coded character that indicates the type of error encountered. The following table is a concise listing of the commands and status:

TO	BOTH	FROM	
		'u'	Waiting for initial command
CTR A			Start in local mode
'#'			Start in remote mode
	'a'		Acknowledge message
		'b'	Busy

'c'			Clear
'#'			Restart in remote mode
'd'			Lock micro where it is
		'e'	Error [followed by error code (ASCII 0 - 9)]
'F'			Move to aligned home position
'g'			Move slide - till touch
'h'			Move to home position
'H'			Move to home position - instructed amount
	'i'		Inquiry
'j'			Move to alternate home position
'J'			Move to alternate home position + instructed amount
'k'			Move slide + till touch
'L'			Move to aligned alternate home position
'm'			Search for hole [followed by strategy (ASCII 0 - 9)]
	'n'		Not understood
'p'			Push part in collet - direction
'P'			Push part in collet - direction with extra testing
'q'			Push part in collet + direction
'Q'			Push part in collet + direction with extra testing
		'r'	Ready
'r'			Relax manipulator
'R'			Set zero force on first 4 degrees of freedom
's'			Status request
't'			Relax and record for next position
'u'			Push hard in - direction
		'u'	Waiting for command
'v'			Push hard in + direction
'U'			Push hard in - direction with Jitter
'V'			Push hard in + direction with Jitter
'w'			Move slide to home position
'x'			Move slide to alternate home position
'y'			Align angles for insertion in positive direction
'Y'			Align angles for insertion in negative direction
'z'			Emergency stop (Lock micro where it is)

Errors will be ASCII digits 0 to 9

1	Micro didn't move on push (into collet)
7	No part (micro couldn't) touch off

A.6 PROGRAMMABLE STOP

All commands and status exchanged between the TWS controller and the Programmable Stop controller are delimited by a carriage return (CR). The responses "b" and "r" indicate that the programmable stop is busy executing and finished, respectively. The following is a list of commands and status:

ec <0 or 1> CR   Echo

This command sets or resets the ECHO flag. When the ECHO flag is set (1), which it would be for local operation, the command sent to the programmable stop is echoed back to the terminal screen, with a LF (line feed) character. When reset (0), there is no echo or LF. This is the normal power up state, and the workstation controller does not normally need to use this command. The response to this command is a "b CR" upon receipt and a "r CR" after completion.

gh CR   Go Home

This command sets a flag which causes the stop to move back to the home limit switch and initialize home position. The response to this command is a "b CR" upon receipt and a "r CR" after completion.

gl <0 to 535> CR   Go to Location < >

This command moves the stop to the absolute position specified by the number. The positions are referenced to home position which is 0. The stop can travel either forward or back to reach the position. A unit length of travel is approximately 0.014 inches. The response to this command is a "b CR" upon receipt and a "r CR" after completion.

sd <0 or 1> CR   Shutdown the Motor

If a 1 is entered, this command sets a flag which deactivates the output stage of the motor driver, disabling the motor. If a zero is entered, the motor is enabled. The response to this command is a "b CR" upon receipt and a "r CR" after completion.

st CR   Start

This command is used as a software reset. It reinitializes all the variables and registers, and sends the stop back to reset the home position. The response to this command is a "r CR".

In terms of programmable stop responses, there are four error messages that can be sent back to the workstation controller. The messages have the format "e <number> CR" , where <number> is the error type. The quotation marks and spaces are not included.

One error condition is an unrecognizable command. This usually occurs when the command is misspelled. Another occurs when an illegal numeric parameter is used, for example, a number greater than 535 with the "gl" (Go to Location) command. A third error condition occurs if the home limit switch is activated without the command to do so having been given. The fourth condition is if the overtravel limit switch is ever activated.

A.7 TURNTABLE

All commands and status exchanged between the TWS controller and the turntable controller are delimited by a carriage return (CR). The responses "b" and "r" indicate that the Turntable is busy executing and finished, respectively. The following is a list of commands and status:

ec <0 or 1> CR   Echo

This command sets or resets the ECHO flag. When the ECHO flag is set (1), which it would be for local operation, the command sent to the Turntable is echoed back to the terminal screen with a LF (line feed) character. When reset (0), there is no echo or LF. This is the normal power up state, and the workstation controller does not normally need to use this command. The response to this command is a "b CR" upon receipt and a "r CR" after completion.

gl <0 through 2047> CR   Go to Location < >

This command moves the platter to the location specified by the number. The proper piston for this location is automatically determined. This command is to be used with the "xp" and "lp" commands for raising and lowering the pistons. The response to this command is a "b CR" upon receipt and a "r CR" after completion.

xp CR   Extend Shaft

This command is used only in conjunction with the "gl< >" command. It causes the shaft which was defined by the latest use of the "gl< >" command to be raised. The response to this command is a "b CR" upon receipt and a "r CR" after completion.

lp CR   Lower Shaft

This command is used only in conjunction with the "gl< >" command. It causes the shaft which was defined by the latest use of the "gl< >" command to be lowered. The response to this command is a "b CR" upon receipt and a "r CR" after completion.

sd <0 or 1> CR   Shutdown the Motor

If a 1 is entered, this command sets a flag which deactivates the output stage of the motor driver, disabling the motor. If a zero is entered, the motor is enabled. The response to this command is a "b CR" upon receipt and a "r CR" after completion.

st CR Start

This command is used as a software reset. It reinitializes all the variables and registers and disables the motor drive. The response to this command is a "r CR".

There are four possible error messages that can be sent back to the workstation controller. The messages have the format "e <number> CR" , where <number> is the error type. The quotes and spaces are not included.

One error condition is an unrecognizable command. This usually occurs when the command is misspelled.

Another occurs when an illegal numeric parameter is used, for example, a number greater than 66 with the "gl" (Go to Location) command. A third error condition is when the platter is commanded to move with a piston still raised, and a fourth occurs when a raise piston command is given when the platter is still in motion.

REFERENCES

1. Simpson, J.A., Hocken, R.J., Albus, J.S., "The Automated Manufacturing Research Facility of the National Bureau of Standards" Journal of Manufacturing Systems, Society of Manufacturing Engineers. 1982, Volume 1, pp. 17-32.

2. Hocken, R., Nanzetta, P., "Research in Automated Manufacturing at NBS", Manufacturing Engineering, October 1983, Volume 91, No. 4, pp. 68-69.

3. Nanzetta, P., "Update: NBS Research Facility Addresses Problems in Setups for Small Batch Manufacturing", Industrial Engineering, June 1984, pp. 68-73.

4. Wolf, W., Magadanz, P., "Block Tool System Fulfills the Tooling Requirement for Unattended Turning", Carbide and Tool Journal,, May-June 1984, Volume 16, No. 3, pp. 11-17.

5. McNally, Paul F., "Motion Control in Loading Task", Robotics World, March 1985.

6. Lyle, Charles S., "A Complete Tooling System for the Multifunction Flexible Turning Machine", Advanced Manufacturing Technology for Cells and FMS, February 25-27 1986, MR86-127, pp. 1-12.

7. Albus, J.S., Barbara, A.J., Nagel, R.N., "Theory and Practice of Hierarchical Control", Proceedings of 23rd IEEE Computer Society International Conference, September 1981, pp. 18-39.

8. Donmez, M. Alkan, Lee, Kang B., Liu, Richard C., Barash, Moshe M., "A Real-time Error Compensation System for a Computerized Numerical Control Turning Center", Proceedings of 1986 IEEE International Conference on Robotics and Automation, April 1986, Volume 1, pp. 172-176.

9. Donmez, M. Alkan, Blomquist, D.S., Hocken, R.J., Liu, Richard C., Barash, Moshe M., "A General Methodology for Machine Tool Accuracy Enhancement by Error Compensation", Precision Engineering, October 1986, Volume 8, Number 4.

10. Donmez, M. Alkan, Liu, Richard C., Barash, Moshe M., "A Generalized Mathematical Model for Machine Tool Errors", paper presented at the 1986 ASME Winter Annual Meeting, PED-Volume 23, 1986, Symposium on Sensors and Controls for Manufacturing.

11. Slocum, Alexander H., "Mechanical Design of a Five Axis Robotic Fine Positioning Device", paper presented at the 1986 ASME Winter Annual Meeting, Symposium on Integrated Intelligent Manufacturing Design.

12. Bandy, H. T., Carew, V. E., Boudreaux, J. C., "An AMPLE Version 0.1 Prototype: The HWS Implementation", to be published as an NBSIR.

13. Boudreaux, J. C., "The AMPLE Project", NBSIR 86-3496, March 1987.

14. Peris, James P., "Development of an Automatic Tool Setting System for NC Turning Centers", the paper is being prepared for publication.

GLOSSARY

A

AMPLE:	Automated Manufacturing Programming Language Environment
AMRF:	Automated Manufacturing Research Facility

C

checksum:	The sum of the ASCII hexadecimal values in a block of text used to confirm the validity of the text transmitted.
command mailgram:	The command block given by the cell controller.
common memory:	A communication system which emphasizes the independence of its readers and writers. Memory locations are accessible to workstations, the cell controller, and the database.

D

device level:	The lowest level in the TWS controller hierarchy; the level where equipment operates.
DML:	A data manipulation language for declaring the executable statements on the application database.
DML request:	A DML command used to obtain required AMRF database reports as needed by the workstation containing a DML string.
DML string:	A variable length string written in DML which directs a database operation to be performed.

E

end-effector:	The work performing device at the end of the robot arm.
equipment instruction list:	The list of instructions for the workstation devices derived from a process plan.
Equipment Item Action report:	A report used to change the location of an inventory item at the equipment level. In the TWS, this report is also used to update the current name of the part as it is being made to the cell controller; spelled as EQUIP_ITEM_ACTION in AMRF database transactions.

G

gripper: The robot end-effector.

I

integrated mode: Automatic operation of the workstation in conjunction with the AMRF cell controller and database.

J

Job level: The highest level in the TWS controller hierarchy. The level of the workstation controller which manages tasks, decomposes process plans, and acts as the interface between the workstation operator and the workstation controller.

L

local mode: The control of a device from a dumb terminal.

Lot Status report: This report describes the parts of the lot (by serial number) and the identifiers that constitute a lot as specified in the cell commands; spelled as LOT_STATUS in AMRF database transactions.

LVDT: Linear Variable Differential Transformer, a linear position measuring device.

M

mailbox: The network view of the common memory locations used for the access of commands, statuses, and data.

MAP: Manufacturing Automation Protocol.

N

Network common memory: The AMRF common memory.

P

PL/M or PLM: Programming Language for Microprocessors.

procedure: In PLM, a set of statements which define an algorithm.

R

RCI: a microprocessor-based robot controller interface; it enables the workstation controller to remotely control the robot.

remote mode: Control of a device from the workstation controller.

S

stand-alone mode: Automatic operation of the workstation independent of the AMRF cell controller and the database; batch commands are entered by the workstation operator.

status mailgram: The status block returned to the cell controller.

T

task level: The second highest level in the TWS controller hierarchy; the level where work elements are executed, e.g. GET_BLANK, LOAD_BLANK, etc.

TOP: Technical Office Protocols.

Tray Contents report: A report from the AMRF database that specifies the item serial number for the parts located in a section of a tray specified in a cell command; spelled as TRAY_CONTENTS in AMRF database transactions.

Tray Definition report: A report from the AMRF database that describes the tray layout of the tray type specified in the cell command; spelled as TRAY_DEFINITION in AMRF database transactions.

TWS: Turning Workstation.

TWSC: Turning Workstation Controller.

W

Work element: A list of device commands that comprise a task.

Workstation: The assemblage of machine tools, robots, buffering systems, and computers used to produce parts.

Workstation Item Action report: This report is used to indicate that a part's location has changed and may be taken over by another equipment level component (e.g., the robot cart); spelled as WS_ITEM_ACTION in AMRF database transactions.

INDEX

NBS-114A (REV. 2-80)

U.S. DEPT. OF COMM. **BIBLIOGRAPHIC DATA SHEET** *(See instructions)*	1. PUBLICATION OR REPORT NO. NBSIR 88-3749	2. Performing Organ. Report No.	3. Publication Date APRIL 1988

4. TITLE AND SUBTITLE

The Turning Workstation in the AMRF.

5. AUTHOR(S) Alkan Donmez, Robert Gavin, Lew Greenspan, Kang Lee, Vincent Lee, Eric Reisenauer, James Peris, Charles Shoemaker, Charles Yang.

6. PERFORMING ORGANIZATION *(If joint or other than NBS, see instructions)*

NATIONAL BUREAU OF STANDARDS
U.S. DEPARTMENT OF COMMERCE
GAITHERSBURG, MD 20899

7. Contract/Grant No.

8. Type of Report & Period Covered

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS *(Street, City, State, ZIP)*

Navy Manufacturing Technology Program.

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT *(A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)*

The Turning Workstation is a flexible manufacturing workstation developed in the Automated Manufacturing Research Facility (A.M.R.F.) at the National Bureau of Standards. The development of the workstation addressed some of the problems associated with an unattended turning operation which include tool changing, collet loading, collet changing, flexible robot end-effectors, and machine malfunction detection. This document describes the components of the Turning Workstation and its relationship to the A.M.R.F.

12. KEY WORDS *(Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)*

Automation, Collet Changing, Collet Loading, Flexible Manufacturing, Gripper, Integration, Malfunction Detection, Process Plan, Robotic, Turning, Workstation, Workstation Control.

13. AVAILABILITY

[✓ Unlimited]
☐ For Official Distribution. Do Not Release to NTIS
☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.
[✓ Order From National Technical Information Service (NTIS), Springfield, VA. 22161]

14. NO. OF PRINTED PAGES

206

15. Price

$24.95