119
112

DEVICE DRIVER DEVELOPMENT AND IMPLEMENTATION FOR WORK CELL CONTROL

by

Aditya Guleri

Project submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree
Master of Engineering
in
Industrial Engineering and Operations Research

APPROVED:

Dr. M. P. Deisenroth (Chairman)

Dr. H. J. Freeman

Dr. C. E. Nunnally

December, 1988
Blacksburg, Virginia

2

LD
5655
V855
1988
G843
c.2

DEVICE DRIVER DEVELOPMENT AND IMPLEMENTATION

FOR WORK CELL CONTROL

by

Aditya Guleri

Dr. M. P. Deisenroth, Chairman

Industrial Engineering and Operations Research

(ABSTRACT)

Industry's move towards automated manufacturing has been rapid during this decade. In a typical flexible manufacturing environment, a variety of automated manufacturing equipment is linked together with a communication network and a part transportation system. Since vendors of automation equipment have no communication standards to adhere to, integrating these machines becomes a problem.

This project addressed the problem by creating a software interface between robots, CNC machines and a workcell controller. The library of functions created for the Dyna CNC machine and the IBM 7545/7547 robots will aid the future user to create software for the workcell. By using the library functions, a user will be insulated from the lower level functioning of the machines and need only be concerned about the operation of these functions.

ACKNOWLEDGEMENTS

This research is dedicated to Shama and Suresh Guleri, my parents, without whose love, support and encouragement, I could not have accomplished what I have.

To Dr. M.P. Deisenroth, my advisor and my teacher, my sincere thanks and deepest gratitude. The experience of working with him was an honor and shall remain one of my cherished memories.

Thanks is also due to Dr. C. E. Nunnally and Dr. H. J. Freeman for serving on my advisory committee. Their criticism of the original draft of this project influenced the final form considerably.

This project would not have seen its completion without the support and help of Bart Moore, Chell Roberts, Ed DeMeter, Joni Chambers, Jai Saboo and Satyajit Mecker. Their efforts are sincerely appreciated.

I would like to thank Radhika Guleri, Jyotshana Sequeira, Horace Sequeira, Vincent Miranda, Shireen Jayatilaka, Michelle Jayatilika, Sandeep Bidani, Kimberley Burrows and Paul Kallan for their love and support. A

special thanks to Jyotshana for keeping the dragon alive.

To Priya Ramamurthy, my friend, for being there for me through high and low. I can't thank you enough.

TABLE OF CONTENTS

Page

1. **INTRODUCTION** 1

2. **PROBLEM STATEMENT** 3

3. **LITERATURE REVIEW** 6

4. **OBJECTIVE** 14

5. **METHOD OF APPROACH** 18

6. **SYSTEM DESCRIPTION** 20

SYSTEM HARDWARE 20
- CIM laboratory 20
- IBM 7545/7547 industrial robots 22
- Dyna 2200/2400 CNC milling machine 24
- Computer network 26
- Machining cell hardware 28

SYSTEM SOFTWARE 29
- Milling machine 29
- IBM 7545/7547 robot 30

7. **MILLING FUNCTION LIBRARY** 31

- INITIALIZE() 35
- DOWNLOAD() 37
- ENDLOAD() 39
- STATUS() 40
- INIT_COM() 42
- GET_COM_STATUS() 42
- SEND_COM() 42
- RECEIVECOM() 43

8. **ROBOT FUNCTION LIBRARY** 45

- START_CYCLE() 49
- DOWNLOADROB() 51
- STOP_CYCLE() 53
- APPLICATION*() 54
- HALT_CYCLE() 56
- MC_STATUS() 57
- REJECT_STATUS() 59

Page

RETURN_HOME() 61
AUTOSTART() 62

9. **MENU DRIVER FUNCTIONS** 64

10. **CONCLUSIONS AND RECCOMENDATIONS** 72

11. **REFERENCES** 76

12. **APPENDIX** 78

A: Program listing 78
B: Dyna operations 123
C: Examples of Dyna programs 126
D: Example of compiled AML/E program .. 127
E:BIOS_SERIALCOM() 128
F: Communication protocol of IBM 7545/ 7547 robot 132

VITA 135

1. INTRODUCTION

American companies, for most of the early eighties, were neglecting their manufacturing in the US. They are now investigating ways to improve the utilization of their resources in the manufacturing of goods. With the technological advancement in manufacturing equipment such as numerically controlled machines, robots, automatic storage and retrieval systems, conveyor systems, etc., one would be inclined to believe that a solution to the "manufacturing malady" was in sight. However, mechanization of the manufacturing process alone cannot produce the desired results. What is required is a systems approach to the problem: the integration of the various "islands of automation" [4] under a hierarchical span of control.

Functional groups in a corporation may view factory automation from distinct perspectives. Product or process design engineers may see Computer Aided Design (CAD) and Computer Aided Manufacturing (CAM) systems as the most obvious path. And very often, management might be convinced that once an executive decision has been made to automate, all the pieces will fall into place.

Nowadays an acceptable philosophy for successful implementation of Computer Integrated Manufacturing (CIM) is the "top-down-bottom-up-approach". One of the major reasons the planning is done from the top down is because the planners can get a more complete picture of the entire system they plan to integrate. The "bottom-up" part of the strategy entails implementation of the plan formulated by the planners, starting at the lowest level and working upwards. The bottom level, which consists of the machines on the shop floor, constitutes the foundation of the CIM effort and is the first step addressed by the people implementing the system. When it is operational, the next level of the system organization is addressed and this outward spread advances the factory towards a more integrated system of manufacturing control.

2. PROBLEM STATEMENT

Various models of production systems describe the organization of separate modules that, through cooperation, behave as a unified system for the control of the manufacturing process with respect to the factory floor. This gives rise to a hierarchy of levels within the plant for attaining this unified control. At the lowest level, machines become grouped into what are called cells. A flexible manufacturing or assembly cell has been defined as the sum of all the devices required for automatically producing a family of different parts or components on one physical capacity unit [3]. These cells consist of a variety of automated manufacturing equipment linked together with a communication network and a part transportation system. There is a great variety in the type of automated equipment used by manufacturers due to the uniqueness of each production facility. Since vendors of automation equipment have no standard code to which they have to adhere, integrating these machines under a flexible manufacturing environment becomes a problem. Under these conditions a user has an option of either:

1) Obtaining a turnkey system from a vendor, in which case he/she has to rely on the vendor to supply

application expertise. In case the user wants to expand further or has some problems with the system, the vendor may have to be consulted often at a higher cost.

2) Obtaining various off-the-shelf machines and integrating them with in-house expertise. Off-the-shelf components may result in huge developmental efforts in terms of time and money.

Flexible Manufacturing Cells (FMC) must integrate and coordinate a diverse group of machines into one synergistic unit. This synergism has never been achieved to the satisfaction of the manufacturing community, largely because of the communication gaps between different kinds of machines. Manufacturers of automated manufacturing equipment develop the machines more as a stand alone piece of hardware with a dedicated computer than as a part of a system under a supervisory computer. This means that each machine operates on its own protocol and configurations which may be totally different from the machine with which it is to be interfaced in the FMC environment. This "divided we stand, united we fall" modus operandi has made integration of automated equipment a problem and has slowed down the implementation of flexible manufacturing cells in industry.

Another problem found in the creation of manufacturing cells is controlling the interactions of the individual machines given that they are interfaced with and "talk to" each other. Developments in microprocessor technology have led to improvements in the control of individual machines. However, when attempts at controlling more than a few machines at the same time are made, the problem of controlling the interactions of the machines can be overwhelming. By partitioning the control problem into smaller, more manageable and less complex control modules and arranging them in a hierarchical fashion, the control system becomes easier to implement and modify.

3. LITERATURE REVIEW

The advent of sophisticated automated equipment such as robots or CNC machines and the proliferation of computer hardware and software have made the concept of the fully automated and integrated manufacturing system a reality. Industrial acceptance of flexible manufacturing cells can be attributed to several causes. According to Palframan [8], FMCs are cheaper than Flexible Manufacturing Systems (FMS) and thus there is a lower financial risk to the customer. Being less expensive than FMSs, FMCs are more easily justified; FMCs give enhanced throughput, better quality, faster response to changes, better data handling and a finer control on the production. Another advantage of FMCs is that there is no need for a very large centralized computer system. This may not reduce the complexity of the system, but it makes it easier to manage and makes the system a lot more robust.

In an attempt to address the problem of controlling machine interaction, a lot of research has been done on generic factory models upon which to base systems design and the interface between systems. A standard factory model must address all of the necessary functional, control, data

flow, and interface issues [11]. Furthermore, the model should be based on fundamental scientific principles and be partitioned into submodels that can be readily understood by system developers. These layers of control or hierarchical levels vary depending upon the task to be performed. As is noted in the Flexible Manufacturing Systems Handbook [6], the use of a hierarchical arrangement of system functions is common, but the way the functions are allocated to the various control levels and the way they are divided varies from installation to installation.

Limited research has been done in the area of cell controller design. The DEC / Philips [9] model of a controller has access to local/ global data and commands from the controller above it. The model issues commands to the subordinate controllers and receives sensory input from them. The DEC/ Philips model of a controller is made up of components performing three functions and they are interconnected as illustrated in Figure 1:

1) goal or task decomposition (H);
2) sensory data processing (G);
3) world model data representation, storage, and retrieval (M).

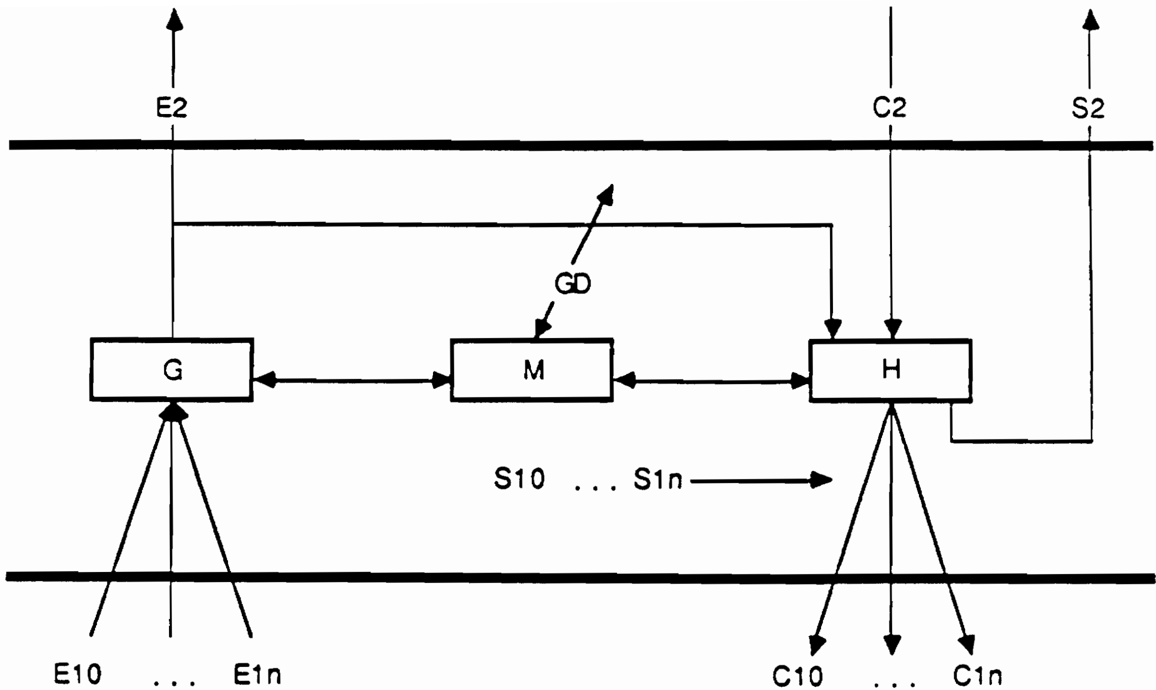


Figure 1: Model of the DEC / Philips Controller [9]

The H function receives a command from the controller above it, for example, C2, to perform a certain function. The H function decomposes C2 into subcommands, C10 to C1n, based on the current state of the system under the domain of the controller. C2 has been fulfilled when all subcommands C10-C1n have been executed by the subordinate controllers. Status information concerning the execution of the

subcommands, S10-S1n, is returned to H. Sensory data from various sensors, E10-E1n, concerning the conditions of the environment under the domain of the controller is processed by G. Relevant information is passed to H to aid in decision making. The M function is the controller's view of its environment. It contains such information as the capabilities of the subordinate controllers. It does not however, concern itself with the actual working of the subordinates. M has access to both local data and global data (GD). GD is information about part of the system which is outside the domain of a particular controller but which may influence the decision making in some way. The Reference Model of Production Systems [9] stresses that the H, G, and M components of the general model are abstract concepts and do not necessarily represent three separate physical systems. In fact, the implementation of the H, G and M components at one level of the hierarchy may differ from the implementation of the H, G and M components at another level.

The National Bureau of Standards (NBS) is one of the major proponents for the use of hierarchical control and is striving to develop standards in this area through its work in the Automated Manufacturing Research Facility (AMRF). The AMRF has five levels in the test-bed control hierarchy: facility, shop, cell, workstation and equipment level. A

level of the AMRF hierarchical control structure is a layer that may be expected to execute several functions in parallel to ensure that the jobs assigned by its supervisor are successfully completed. These functions are:

1) Decomposing complex tasks into simpler subtasks
2) Assigning subtasks to appropriate subordinates
3) Allocating required resources to subordinates
4) Monitoring the execution of subtasks
5) Updating schedules to reflect changes in delays

At the lowest level, the Equipment Control Systems are "front end" systems that are directly connected to commercial equipment or industrial machinery on the shop floor. The equipment control system interfaces to a Workstation Control System above and directly to the vendor-supplied controller on the hardware under its control. This approach implies that it may be possible to partition any equipment control system into a high-level controller, which is hardware independent, to perform the command decomposition and a low-level controller, which is hardware dependent, to monitor task execution [7].

Rather than take the NBS or the DEC/Philips approach and develop an entirely new philosophy for work cell control, other researchers have sought to improve and adapt existing strategies to counter the current control problems. Due to the expense of developing custom control software for

production systems, Arai, Hata, Imakubo, and Kikuchi [1] suggest a standard, two-level workcell control system. The lower level controller is responsible for the actions of up to four NC machines. The upper level controller contains all production information and transmits the required NC programs to the lower level controller. Eight functions are performed by this system: data management, schedule running, machine control, production record gathering, interactive quality control, failure diagnosis, machine monitoring and material handling control.

Bertrand [2] seeks to extend the MRP-II philosophy of production control system design by developing a methodology for designing hierarchically structured control systems. Using the definition of a production unit as a collection of capacities, operations, and materials for performing a certain task, he divides the production control problem into three hierarchically arranged procedures: 1) periodically generate a Master Plan based on the actual state of the system and the anticipated future states; 2) during a period, generate inputs to the various production units that will coordinate the efforts of the production units in the fulfillment of the Master Plan; and 3) control the actual inputs to the production units based on the state of each production unit, the coordination inputs, and the requirements of the system.

There have been several publications on the design of generic factory models and control hierarchies but limited research on addressing the problem of integrating the plethora of automated equipment made by the various vendors in the market. Manufacturing Automation Protocol (MAP) is a bold attempt by General Motor's to force a standard on the products made by different vendors but the arm twisting has been successful only within limits. Since MAP is a concept and its practical implementation is still in a nascent state the aforementioned problem of integration still exists. MAP is used for all communications between planning and scheduling functions and the various components of the factory control system. In a typical setup, MAP is used [10]:

1) To pass commands from factory control to the cell controller and the material handling controller, and to receive status and monitoring data.
2) To upload and download device programs from and to the programmable devices at the manufacturing cells.
3) To transfer factory status and schedule data between the factory control system and the factory scheduling system.
4) To handle communication among the various functions within the factory control system.

As mentioned before, the acceptance of standards as dictated by MAP have been partially implemented by the vendors in the manufacturing of their hardware but they form a small percentage of the lot. Thus, as specific devices in an implementation may not support MAP, hardware choice usually establishes fixed constraints. Constraints also arise from the use of software packages for specific areas in the implementation. Therefore, MAP has not been able to solve the problem of integrating automated equipment completely.

It is evident from the literature reviewed that attempts are being made to solve the problem of integrating automated equipment. But, in reality, this problem still exists for people in industry trying to implement computer integrated manufacturing. The Computer Integrated Manufacturing Laboratory at Virginia Tech uses equipment made by different vendors and integrating them was a challanging task. This project proposed a solution for the integration of these different machines.

4. OBJECTIVE

The Computer Integrated Manufacturing Laboratory in the Industrial Engineering and Operations Research Department of Virginia Tech was planned to provide a facility for instruction and research in the integration and control aspects of computer-based manufacturing technologies. As planned, it will contain a CIM system consisting of two three-axis CNC milling machines, two industrial robots, an automatic storage and retrieval system, a machine vision system, a material delivery system, a programmable controller and a network of personal computers. The architecture of the hierarchical control has been based on the intended structure of the CIM Laboratories control software. It will consist of three basic levels - system, cell, and equipment levels.

The system level control facilities are responsible for the overall performance of the system. The system controller coordinates the production and support activities that are carried out by the cell controllers at the next lower level. Its planning horizon can be from a few hours to several days. The system-level controller is above the cell controller in the control hierarchy. Two major modules

have been identified within the system-level controller, i.e., a task manager and a resource manager. The task manager does capacity planning, identifies production resource requirements, summarizes quality performance data, generates schedules, tracks individual orders to completion and tracks equipment utilization. The resource manager monitors and updates levels of all raw material stock and replacement parts inventory necessary to run the factory. Based on the availability of resources and the tasks that need to be performed for the completion of each batch, the system controller sends its commands to the cell controller by posting it in a common area in memory called a "mail box". Commands are passed down by the system controller to the cell controllers which sends up status reports.

The cell level is responsible for directing and coordinating the actions of the machinery under its domain. The machining cell controller has an IBM 7545 robot and two DYNA 2400 bench top CNC milling machines as part of its domain. The cell controller will be responsible for getting directions from the system controller by reading them from the "mail-box", breaking the directions down into smaller modules and executing them on the machinery under the cell controllers domain. To do this the cell controller will be using the functions which have been developed as part of a software "tool-kit". The cell controller will also be

monitoring the machinery for digital input and output (I/O) and for error conditions.

The particular focus of this project will be towards the development of this cell level controller. It will be addressing two areas:

1) To develop a "tool-kit" of functions in the C programming language for the control of the machinery under the domain of the cell controller.

2) To integrate machine actions into a cohesive flexible system and to eliminate the dependency on human intervention for monitoring and evaluating the system.

A "tool-kit" is a collection of functions which are used repetitively in the development of control software for any given system. The idea is the same as a tool-box containing several instruments each of which is to be used for a specific purpose. Extending the analogy to the "tool-kit", one can assert that each of the functions is best suited for the job it is meant to do which in turn results in efficient and fast software. The "tool-kit" functions are stand-alone modules, thus future users may upgrade the system by adding/changing machines without having to go through large developmental efforts because they do not have

to bother about the internal working of the functions available in the "tool-kit".

Integration of machines is not so easy in a typical manufacturing cell because each machine has its own protocols, digital input and digital output capabilities, control language etc.; thus the interfacing software has to process a variety of data and present it to the machine controllers in a form that can be understood. This naturally makes the interfacing software extremely machine dependent. But by using a "tool-kit" that has functions that can handle the variations in going from machine to machine, the writing of interfacing software becomes a very structured and visible task. The need for human intervention in classical manufacturing environments arises because ongoing processes create situations where decisions need to be taken in order for the process to continue. But in FMCs error recognition, error correction, system resetting etc. in an integrated cell are done by the software controlling the cell.

The third level of control is the equipment control level. It consists of robot and NC machine controllers, digital I/O equipment linked to sensors and actuators. The functions are carried out by the microprocessors resident in the equipment.

5. METHOD OF APPROACH

This project was divided into two phases. The first phase was the development of the aforementioned software "tool-kit" for controlling /monitoring the machines under the domain of the cell controller. The second phase entailed demonstrating use of the tool-kit as applied to the work cell.

The development and implementation was done on a DOS based IBM AT in the C programming language. The language has been selected for use because of its capability for fast, efficient processing and producing compact code. Additionally, C is a middle level language whose strength lies in the user being able to write high level mathematical functions while still being able to access lower level functions. Being a middle level language, it exhibits some of the fine control associated with assembly language and at the same time provides the good programming principles normally associated with structured languages like FORTRAN and PASCAL. It also allows easy access to bit level manipulation unlike higher level languages.

The second phase of the project was the demonstration

of the use of these routines by the creation of a menu driven cell controller. This program can be used to provide keyboard control of the various subsystems within the machining cell. Operator inputs cause the execution of the appropriate tool-kit functions in order to accomplish desired actions of the work cell equipment. This program can be used for "manual control" of the work cell as well as software system testing and debugging.

The software created as part of this project was aimed at:

1) showing the operation of the "tool-box" functions.
2) permitting keyboard control of the work cell mechanisms.

6. SYSTEM DESCRIPTION

SYSTEM HARDWARE

Computer Integrated Manufacturing Laboratory

The Computer Integrated Manufacturing(CIM) Laboratory was developed to create a laboratory environment where research and instruction in the integration/control aspects of computer based manufacturing can be facilitated. The components of the CIM laboratory are given in Figure 2. Specifically included in the laboratory are:

1. Two 3-axis milling machines
2. IBM 7545 robot (machining sub-cell)
3. IBM 7547 robot (assembly sub-cell)
4. Shuttleworth conveying system
5. Automatic Storage and Retrieval System (AS/RS)
6. GE Optovision II system
7. TI 565 programmable controller
8. Sensors and actuators
9. Network of Personal computers.

The system was designed to produce two initial products, a wax SCARA robot and a wax toy milling machine. All parts for the specific products requiring machining are

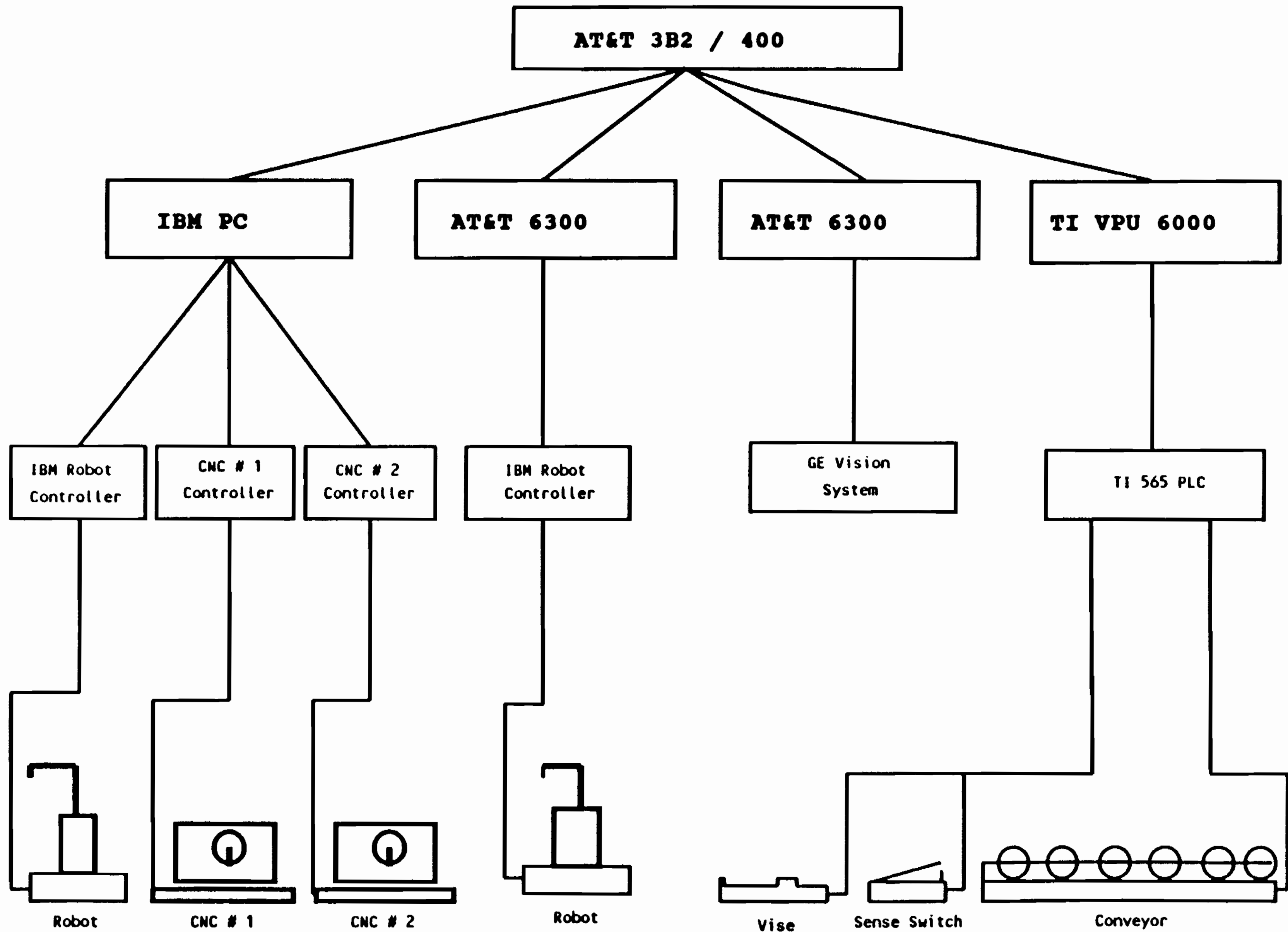


Figure 2: Overall structure of the CIM laboratory

assembled onto a pallet at the first work cell. The pallets are then transported by the conveyor to the second work cell where the robot loads the CNC milling machine according to the machining program for a specific product. When machining is complete, the pallets are sent back to storage or directly to the assembly work cell for assembly of the product. Completed pallets are returned to storage.

A major portion of the activity in the CIM laboratory is done by the robot and the CNC milling machines. The following sections describe the robot and the CNC milling machine in greater detail.

IBM 7545/7547 industrial robots

The IBM 7545/7547 industrial robot is classified as a continuous path Selective Compliance Assembly Robot Arm (SCARA) robot. There are three main components of the robot: the manipulator, the controller, and the control panel.

The manipulator is a two-jointed arm structure with four degrees of freedom. The joints of the arm provides 2 degrees of freedom through their swivel motion. The end of arm rotation called the roll axis, provides the third degree of freedom. The end of arm also provides a fourth degree of

freedom through the vertical shaft.

The controller contains most of the electronics for control of the manipulator operation. A microprocessor coordinates the manipulator movement and monitors its speed and positioning. The controller receives and stores its control program from the work cell controller, and then drives the manipulator by executing the programs. The application programs instruct the controller as to which ports to monitor, the amount of time to wait for the event, and the type of condition to expect.

The control panel is the operator interface to the controller and the manipulator. The control panel uses pressure sensitive keys, light emitting diodes (LEDs) and an audio feedback device. In a work cell application, the user of the control panel is relegated to getting the manipulator powered-up and on-line. The rest of the functions are performed by the work cell controller.

External devices are synchronized with the manipulator through the use of digital input (DI) and digital output (DO) ports. These ports monitor switch closures (DI) external to the system or close relays (DO), allowing events to occur.

Communications to the controller is accomplished by using asynchronous communication line protocol. All data is sent as 7 bits (plus parity) American National Standard Code for Information Interchange (ASCII), even parity, at 4800 baud (bits per second), with two stop bits. An Electronic Industry Association (EIA) Recommended Standard (RS)-232C interface communicates with the work cell controller.

Dyna 2400 CNC milling machine

The DYNA 2400 is a light duty bench top,three axis vertical CNC milling machine. The machine requires a single phase power source of 120V AC, 60 Hz. The power requirement is 500 watts. The machine is generally operated from the controller keyboard. The range of the spindle is 0-10,000 RPM and can be achieved in three steps by a change in the position of the spindle drive belt. The spindle can be turned ON and OFF either manually or by instruction in the program. The entire spindle head can be lowered, elevated and rotated on the head post.

The fourth axis on the Dyna machine is an optional rotational table that can be mounted horizontally or vertically. For machining operations, the X and Y table, and the spindle head (Z axis) move, but for convenience, it is assumed that the tool tip moves (since it is the actual

cutting point). The three axis X, Y, and Z are mutually perpendicular to each other. Their intersection point is defined as the zero coordinate and is expressed as (0,0,0).

Three different zero coordinate systems are associated with the machine. The Home zero is the position when the X,Y and Z axes are physically at their limit switches. The Reference zero is the position which is established manually by the operator and serves as a reference point for tool motion. The Local zero is specified by the user within the program and is used mainly for circular motion and to avoid unnecessary calculations of the coordinate data.

The DYNA milling machine allows the user to specify the feedrate in the X,Y and Z directions. The feedrate specified should lie between 0.1 and 32.0 inches per minute. If the feedrate is not specified , a default value of 8.0 inches per minute (203.2 mm/min) is assumed.

The DYNA machine has four operating modes. The manual mode used for the tool calibration and for manual operation of the machine. The line number mode is used for entering the program into the controller of the machine. Programs are entered in the program enter mode. The run mode is used for the operating of the machine under program control.

The DYNA machine has its own language, DYNALAN, which consists of a few very powerful instructions which are designed to simplify machine move programming. Along with this there are some functions which are preprogrammed into the basic machine controller. Some of these special functions include mill, frame, rectangular frame, rectangular pocket, circular pocket, arcframe and drill.

The Dyna machine has an RS232 interface which allows the user to interface the controller with an external computer or peripheral communication link. This link may be used for four operations: program uploading, program downloading, line execution and program loading and execution. Program uploading transfers a program from the controller to the computer (to be stored or printed out). The program load option transfers a program from the computer to the controller while the line execute option executes a program from the computer to the controller a line at a time. This allows infinite length but the program execution is slow due to the communication overhead. The program execute option allows the computer to download blocks of program at the time of execution.

Computer Network

The Computer Integrated Manufacturing Laboratory has a

network of computers that control its functioning. Figure 2 shows the overall structure of the computer network in the CIM laboratory. The network contains a number of computers made by different manufacturers. This has been done to make the computer integrated manufacturing effort more generic rather than machine specific.

At the cell control level, the machining cell has an IBM PC as the cell controller. The robot and the two milling machines are connected to it. The cell controller of the assembly cell is an AT&T 6300 PC which is connected to the IBM 7547 industrial robot. The inspection station also has an AT&T 6300 PC which is connected to the GE Vision system. The TI VPU 6000 computer is the host computer of the TI 565 programmable controller (P/C). In reality, as each of the cells have sensors and actuators tied in to the programmable controller, the P/C can be thought of as being physically part of each cell controller.

The system level is responsible for the overall operations and coordination of the individual machines. The system controller coordinates the production and support activities that are carried out by the cell controllers at the next lower level. The system controller is a AT&T 3B2/400 mini-computer which is connected to each of the cell controllers via the AT&T STARLAN network.

Machining Cell Hardware

The machining cell is the specific focus of this project. The machining cell consists of two CNC milling machines and an IBM 7545 robot. Jigs, fixtures, grippers and other devices will be added in future implementations to these basic pieces of equipment to accomplish the machining requirements for each product. In order to accomplish the task assigned, each CNC machine must have specific and unique tooling. For example, the raw base of the wax robot to be manufactured contains a relatively large amount of material to be milled; therefore, a large cutter is assigned to CNC 1. CNC 1 also utilizes a jig with large clearances to accommodate the large tool. At CNC 2 less material is to be removed. Therefore, the CNC 2 has a smaller cutter and the jig has small clearances.

For demonstrating the software library created for the Dyna CNC milling machine, COM 2 is connected to one of the milling machines while COM 1 is connected to the robot. The computer controlling these machines is an AT&T 6300 PC with a 20 megabyte hard drive.

SYSTEM SOFTWARE

The software written for the CIM laboratory as part of this project has been the creation of several high level functions for controlling the CNC machine and the IBM robot. These function form a "tool-box" from which the user can choose the various functions to write a cell control program. The lower level communications and protocol of these specific machines is made transparent to the user who is only concerned about the values returned by these "tool-box" functions.

Milling Machine

The Dyna milling machine has several high level library functions. The INITIALIZE() function is used to let the operator establish the origin of the reference coordinate system before downloading the program that contains the motion statements. The DOWNLOAD() function can be called for transmitting the motion statements of a milling program to the CNC controller. The ENDLOAD() function is executed for ending the downloading sequence to the Dyna CNC machine and returns the controller to the manual mode of operation. The STATUS() function gets the current state of the machine and to identifies the occurrence of any special conditions during the operation of the milling machine.

IBM 7545/7547 robot

There are several high level function in the robot library. The DOWNLOADROB() function is used for transmitting the compiled AML/E program to the robot controller. STARTCYCLE() starts the execution of a program from a user specified partition whereas the STOPCYCLE() function stops the execution. APPLICATION*() function is called for choosing the partition specified by the user in the STARTCYCLE() function while HALT_CYCLE() halts the execution of the robot and takes it back to the uninitialized state. MC_STATUS() is used during the operation of the robot to obtain its current state and to identify the occurrence of any special error conditions whereas REJECT_STATUS() reads the reject status of the robot controller. The RETURN_HOME() function returns the robot to the home position and the AUTOSTART() function puts the controller into the auto operation mode.

The functions described above make use of some primitive functions for basic communications and control. A detailed description of these primitive functions has been given in Chapter 7. These primitives aid in communication between devices, help in configuration of the ports, and help identify the status of the communication port.

7. MILLING FUNCTION LIBRARY

The milling library functions permits the usage of the DYNA 2400 CNC machine as part of the cell environment. These functions provide the interface between the CNC controller and the work cell controller computer. The first part of this chapter describes the operational considerations of the milling machine and the functionality of these DYNA functions. The second part presents a detailed description of the code, its parameters and possible future modifications, enhancements and extensions. Appendix A contains a complete source listing of the milling function library.

Machine states are the various modes in which the machine can be found during the course of its operation within the cell. There are four machine states of interest as shown in Figure 3. At the beginning of the operations in the cell, when the power is not applied, the machine is in an uninitialized condition. This is the "cold start" state and is referred to as STATE 0. Operator interaction is required to get the milling machine from this state to the next state. First the operator is required to clear the

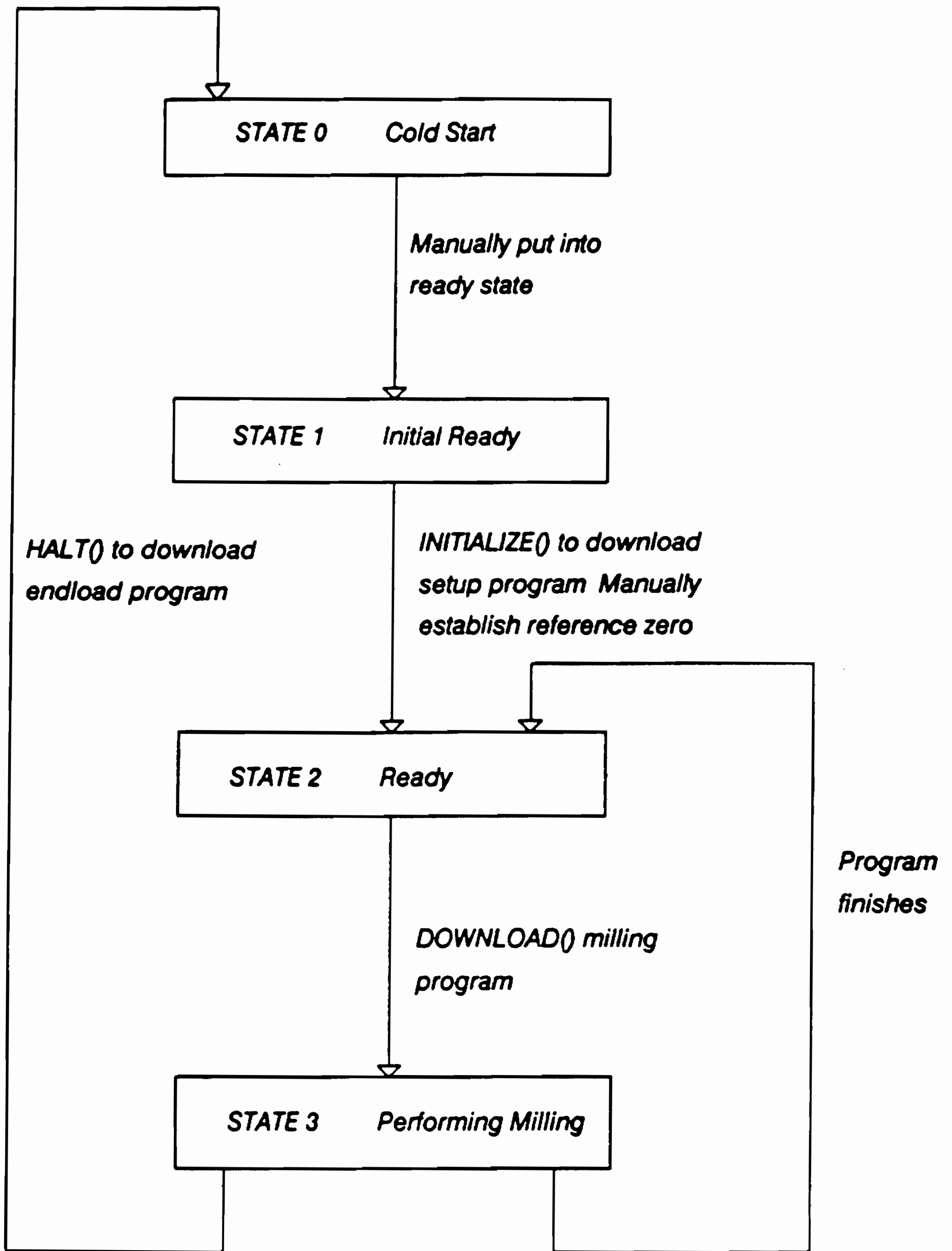


Figure 3: States during the milling operation

memory of the controller and to ready the machine for downloading and automatic execution of a Dyna program. This procedure is explained in Appendix B. When this operation is complete the NC controller indicates the readiness of the NC machine by displaying <READY> on its LCD display and by asserting the RTS line on the RS232 port.

At this point the machine is initialized and ready for a program to be downloaded. While it is possible to load any program at this point, proper work cell operations require the downloading of a special setup program for establishing the reference zero of the machine. This is done by referencing the INITIALIZE() function which downloads the setup program containing the feedrate information for the different axis, the unit of measurement, the tool diameter, and the SETUP command. An example SETUP program is given in Appendix C. At this point the operator has to jog the various axis of the machine in order to set the reference zero of the NC machine.

When this has been completed, the operator depresses the NEXT button on the milling console; the mill then advances to the "ready" state referred to as STATE 2. In STATE 2, the machine is waiting for one of the executable milling programs to be downloaded. These programs contain the Dyna motion statements necessary to machine a desired

part without the initialization block or end code block. Each block contains a SKIP TO command to cause the mill to branch to line 900 of the program. A sample milling program is given in Appendix C. Download of a desired Dyna program is accomplished by referencing the DOWNLOAD() function. The program starts execution as soon as the last line of code has been downloaded.

While the machine is active and executing the downloaded program, it is the milling state referred to as STATE 3. While the Dyna is in STATE 3, the user cannot download another program to the controller nor can the machine be reinitialized. At the end of the execution of the Dyna program, the milling machine once again enters into STATE 2 wherein the user has the option of downloading another block of executable code by referencing the DOWNLOAD() function or ending the program operation using the HALT() function. The use of the HALT() puts the machine back into STATE 0.

During the operation of the Dyna, calling the STATUS() function will return the current state of the Dyna and any special conditions within the states. For example, if the INITIALIZE() function has been referenced, but the operator has not completed the manual setup process, the STATUS() function will return a state of 1 until the operation is

complete.

As explained earlier, the objective of the project was the creation of a library of C functions for communications between the work cell controller and the CNC controller. The functions are further described below and form the core of the "tool-kit" for the NC machine. Additionally, the communication primitives or lower level functions which are referenced by the core library functions are detailed at the end of this chapter.

INITIALIZE (int *ncnumber, char *name, int *ffunction)

The INITIALIZE() function is used to let the operator establish the origin of the reference coordinate system before downloading the program that contains the motion statements. It has three parameters associated with it. The first one, ncnumber, is the number of the CNC machine the user wants to initialize. Valid inputs are either 1 or 2. The second parameter, name, is the name of the initializing program. Different setup programs would be necessary when some of the Dyna program parameters need to be changed. The forcing function, ffunction, is the third parameter and it has two possible values, 1 or 2. Normally the function INITIALIZE() will not allow the user to initialize a machine that is already active, eg. in STATE 1

or 2. However, if there is a machine failure, the work cell controller must initialize a machine it already considers initialized. INITIALIZE() will force initialization, regardless of the machine state, if the forcing function has a value of 2. The forcing function is 1 during normal initialization.

The CNC machine number and forcing function parameters are pointers to integers whereas the name of the initializing program is a pointer to a character. The function starts by checking for the accuracy of the input parameters against the values expected. Additionally, it checks the machine number and the forcing function for values of 1 or 2. Any other value returns an error code to the calling program. Next it checks the value of the global variable COUNT which indicates whether the milling machine has been initialized or not. If the forcing function is 1, ie. normal initialization and the CNC machine has already been initialized, the INITIALIZE() function returns an error code to the calling program. It allows initialization only if COUNT is 0 and forcing function is 1 or the COUNT is 1 and the forcing function is 2.

Next, the function opens the specified initialization file and returns an error code if the file does not exist. If it finds the file, it starts downloading it character by

character using the primitive function SEND_COM(). The Dyna controller requires that each line of code must end with a carriage return/ line feed sequence. Thus, the function appends a Cr [Hex 0A] and Lf [Hex 0D] at the end of each line of code it transmits and at the end of the file it terminates the transmitting by sending a Ctrl Z to the CNC controller. During the entire process of downloading, if an error occurs, the function returns an error code to the calling program. If the downloading process has been successful, it returns an error code of 0 and exits. The six possible error codes are:

0 Initialization was correct

1 Machine already initialized

2 Machine number error

3 Forcing function error

4 Machine not ready

5 Initialization file error

DOWNLOAD(int *ncnumber, char *name)

The DOWNLOAD( ) function is used for transmitting the motion statements of a milling program to the CNC controller. It has two parameters associated with it. The first one is the number of the CNC machine to which the file is to be sent. Valid inputs are 1 and 2 for use in the CIM Laboratory. The second parameter is the name of the Dyna

program which contains the motion statements for the CNC machines. The CNC number parameter is a pointer to integer whereas the name of the initializing program is a pointer to a character.

The function starts by checking for the accuracy of the input parameters against the values expected. The machine number must be 1 or 2. Next this function checks to see if the machine has already been initialized and is not in the active state. This is done by checking the STATE variable for a value of 1 or 2, if STATE has any other value an appropriate error code is returned to the calling program. As mentioned before, when running the SETUP() function there is need for the operator to manually establish the reference zero of the Dyna. During the time the local zero is not set, the RS-232 Ready To Send (RTS) remains low. This condition is checked in the next line by calling the primitive GET_COM_STATUS() function. Next, the function opens the specified milling file and returns an error code if the file is not present. If the file exists, The DOWNLOAD() function starts transmitting the file character by character using the primitive function SEND_COM().

If an error occurs during the downloading process, the function returns an error code to the calling program. Just before exiting the DOWNLOAD() function, the value of STATE

is set equal to 3. If the downloading process has been successful, it returns the appropriate code and exits. Possible error returns from DOWNLOAD() are:

0 Download was correct

1 Machine not initialized

2 Absolute zero not set/ Machine executing program

3 Machine number error

4 File not found

5 Not a valid text file (*future implementation)

ENDLOAD(int *ncnumber)

The ENDLOAD() function is used for ending the downloading sequence to Dyna CNC machine and returns the controller to the manual mode of operation. It has one parameter associated with it. The ncnumber indicates the CNC machine that is to be taken out of the automatic mode. Valid inputs are 1 and 2 for use in the CIM Laboratory. The CNC number parameter is a pointer to an integer. The function starts by checking for the accuracy of the parameter passed down to it against the value expected. The name of the text file used for terminating has been declared as an external since it is common to both of the machines. The function opens the specified ending file and returns an error code if the file is not present. If it finds the file, it starts downloading it character by character using

the primitive SEND_COM(). If an error occurs during the downloading process, the function returns an error code to the calling function. If the downloading process has been successful, it returns the appropriate code and exits. Error codes that can be returned by ENDLOAD() are:

0 Ending was correct

1 Machine code error

2 Could not open file

STATUS(int *ncnumber)

The STATUS() function is used during the operation of the CNC machine to obtain the current state of the machine and to identify the occurrence of any special conditions. The function expects to be passed a pointer to the CNC machine number whose status is being polled. In the present application, valid inputs are either 1 or 2 and any other values will return an error condition.

The Dyna CNC machine can enter into special conditions besides the regular states explained earlier. For example, after running the SETUP() function there is need for manual intervention to establish the location of the reference zero. While this is being done, the RS-232 Ready To Send (RTS) remains low. Additionally, this line remains low when a downloaded program is actively executing. These

conditions are checked in the status function by calling the primitive function GET_COM_STATUS(). The GET_COM_STATUS() returns a byte of information on the status of the chosen communication port. Thus, using a bitwise AND, the function checks whether RTS is high or low. The function checks the STATE at that particular time. If the global variable STATE is 1 and the RTS is low, it indicates that the reference zero has not been set. Else, if the STATE is 2 and the RTS is low, it indicates that the mill is executing a program. Finally, the function itself returns the STATE of the polled CNC machine:

0	Uninitialized, not ready
1	Uninitialized, ready
2	Initialized
3	Downloaded, active
4	Go, active (*future implementation)
5	Incorrect machine code

The functions described above have made use of some primitive functions for basic communication and control. These communication primitives aid in communication between devices, help in configuration of the ports, and help identify the status of the communication port.

INIT_COM(com_port, com_data, com_error)

INIT_COM() is the function used for the initialization of the communication port to which the CNC machine is connected. The parameters are the communication port number and the communication data. The com_data argument is obtained by ORing the constants which specify the data bits, stop bits, parity type and the baud rate for the transmission. The function sets the error flag if the port is out of the specified range. It makes use of the _BIOS_SERIALCOM() function out of the Microsoft C 5.0 BIOS library.

GET_COM_STATUS(comp_port, com_status)

GET_COM_STATUS() is the function used to find the status of the communication port it is polling. It has two parameters associated with it. The com_port is the communication port which has to be polled for status and the other parameter is a pointer to the unsigned integer which returns the status of the communication port being polled. This function also makes use of the _BIOS_SERIALCOM() function. The _BIOS_SERIALCOM() function returns an 8 bit integer which indicate the status of the lines.

SEND_COM(com_port, com_char, com_error)

The SEND_COM() function is used to send characters out of the communication port. It has three parameters associated with it. The com_port is the communication port to which the character has to be sent, com_char is the second parameter and is a pointer to the character which has to be sent. The third parameter is the com_error which too is a pointer to an unsigned integer and in the event of an error, it returns the reason for the error. This function makes use of the _BIOS_SERIALCOM() function from the BIOS library. The BIOS_SERIALCOM() function returns a sixteen bit integer to the SEND_COM function which checks it for states. If bit 15 is set, the data could not be sent and this value is returned in com_error.

RECEIVECOM(com_char, com_error)

The RECEIVE_COM() function is used to receive characters from the specified communication port. It has two parameters associated with it. Com_char is the first parameter and is a pointer to the character which is received. The second parameter is the com_error which too is a pointer to an unsigned integer and in the event of an error, it returns the reason for the error. The function starts by checking the transmission-holding register by using the com_status value of function GET_COM_STATUS().

this until a character is received or an error occurs. This function makes use of the _BIOS_SERIALCOM() function from the BIOS library, which returns a 16 bit integer. The character is returned in the lower byte if the call is successful. If an error occurs, at least one of the high-order bits will be set.

8. ROBOT FUNCTION LIBRARY

The robot functions library permits the control of the IBM 7545 and IBM 7547 robots in a work cell environment. These functions provide the interface between the robot controller and the work cell controller. Since the robots are functionally equivalent from a communications standpoint, the same software library can be used to control either robot. The first part of this chapter describes the operational considerations of the robot and the functionality of these robot functions. The second part presents a detailed description of the code, its parameters, and possible future enhancements and extensions. Appendix A contains a complete source listing of the robot function library.

As mentioned before, in the case of the CNC milling machine, the robot too exists in certain specific states during the course of its operation within the cell. There are three states of interest as shown in Figure 4. At the beginning of the operations in the cell, when the power is not applied, the machine is in an uninitialized condition. It requires operator interaction to activate the manipulator and get the robot on-line. This is the "cold start" state

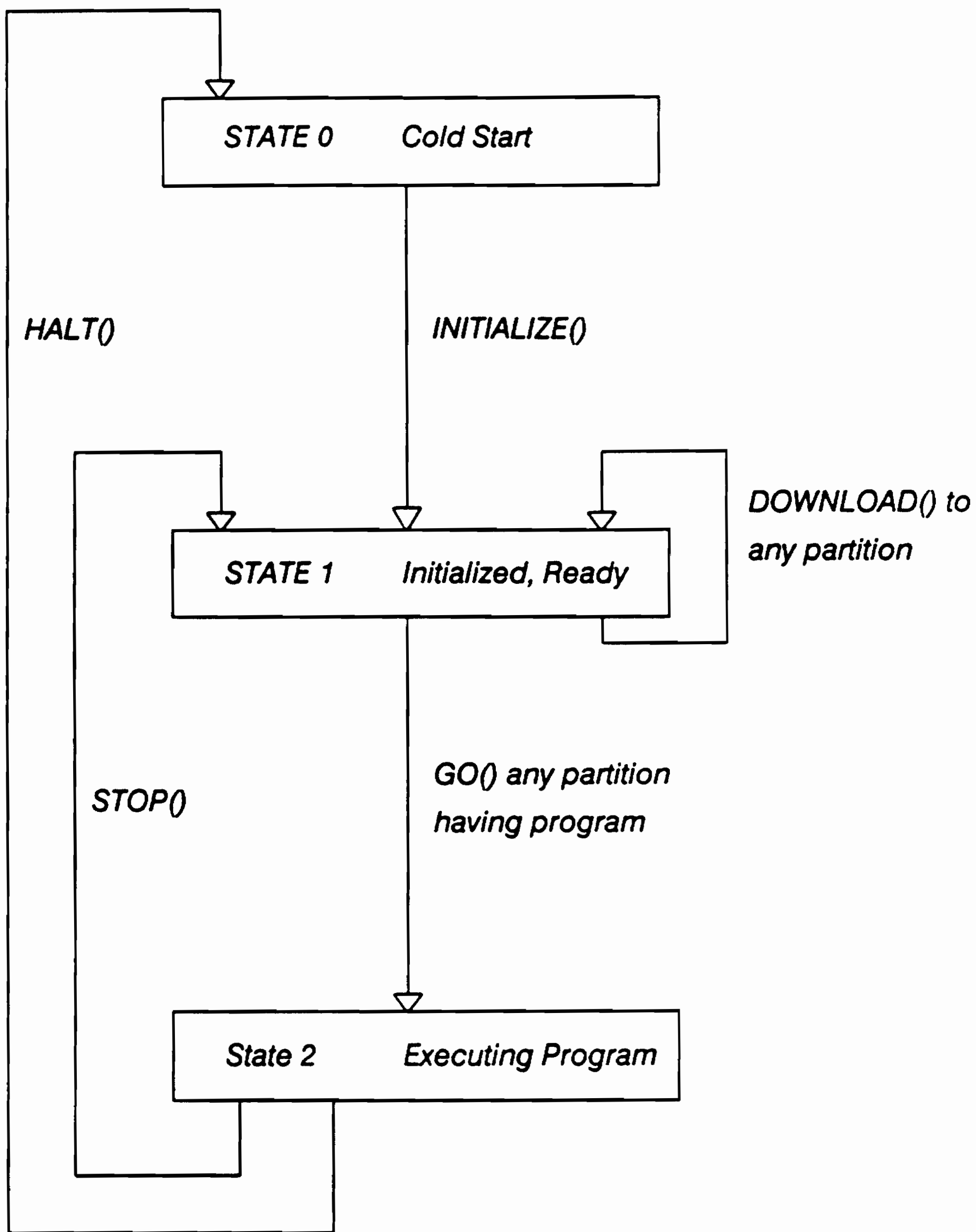


Figure 4: States during the robot operation

and is referred to as STATE 0.

The library function INITIALIZE() is used to initialize the robot. This function returns the robot to its home position and then puts it in the autoexecution mode using primitive functions AUTOSTART() and RETURN_HOME(). This changes the robot to the initialized state referred to as STATE 1.

At this point the machine is initialized and ready for a program to be downloaded. The robot has five memory partitions for storing the programs which are downloaded to it. The downloading of a program to the various partitions is done by using the DOWNLOAD() function. The download function returns to STATE 1 after completing the downloading process. The user has the option of downloading another compiled AML/E program into the robot partitions by using DOWNLOAD() repetitively.

When one or more programs have been downloaded, the user can begin executing the downloaded programs from a user defined partition using the GO() function. The GO() function starts executing the program stored in the chosen partition by using the primitive functions STARTCYCLE() and APPLICATION*(). APPLICATION*() is the primitive that sends the chosen partition number to the controller and

STARTCYCLE() starts executing the program stored in that partition. This "program execution" state is referred to as STATE 2.

The program being executed by the controller can be stopped by using the STOP() function. This brings the controller back to STATE 1. Here again the user may download another program, start a particular application or may choose to stop the entire operation using the HALT() function. The HALT() function changes the STATE from 1 back to 0 and stops the robot from executing any other function since it forces the robot into the uninitialized condition again. The software has been written such that each time the robot is initialized, the user has to download the programs into the partitions because the programs from the previous day are flushed out of the controller memory when the user is exiting a session using the HALT() function.

The STATUS() function is similar to the one found in the CNC milling library. It can be called at any time during the operation of the robot and it returns the current STATE of the robot and also any error code associated with the robot controller or the transmission process.

As mentioned earlier, the objective of the project was the creation of a library of C functions for communications

between the work cell controller and the robot controller. The functions are further described below and form the core of the "tool-kit" for the robot. Additionally, the communication primitives or lower level functions which are referenced by the core library are detailed at the end of this chapter. Appendix F contains a complete listing of the communication protocol followed by the robot controller.

START_CYCLE(int *number, int *partition)

The START_CYCLE() function is used for starting the execution of a program from a user specified partition. It has two parameters associated with it. The first one is the number of the robot which has to be started. Valid inputs are 1 and 2. The second parameter is the number of the partition from which the program has to be executed. Valid inputs are 1 to 5. The robot number parameter, number, and the partition number parameter, partition, are pointers to integers.

The function starts by checking for the accuracy of the input parameters against the values expected. In case there is an error, the function returns an error code to the calling program. Next, the value of the partition parameter is evaluated and call is made to APPLICATION*() function which activates the requisite partition. If an error

condition is returned by the APPLICATION*() function, START_CYCLE() returns an error code to the calling function and exits.

The next block of code deals with the protocol for starting the cycle. The function sends the robot controller an X identifier (hex 58) to indicate the start of an execute transaction using the SENDCOM() primitive function. The controller always responds with an initial Xoff (hex 13). If the controller is able to accept the record an Xon (hex 11) is sent following the Xoff. Both these characters are read by using the primitive function RECEIVECOM(). The START_CYCLE() function then sends the start cycle operand "22" (hex 32, hex 32) to the controller followed by the Cr (hex 0D) and Lf (hex 0A) to indicate the end of the record using SENDCOM(). The controller responds with an Ack (hex 06) to indicate that the record was received correctly, and that the command has completed. START_CYCLE() then sets the value of the global variable which is keeping track of the robot state, STATEROBOT, to 2 and returns to the calling program. In case of an error in the transactions between the robot controller and the START_CYCLE() function, the function returns an error code to the calling function and exits. The possible error codes are:

0	Started cycle
1	Robot number error

2	Starting function error
3	Partition empty

DOWNLOADROB(int *number, int *part, char *name)

The DOWNLOADROB() function is used for transmitting the compiled AML/E program to the robot controller. It has three parameters associated with it. The first one is the number of the robot to which the file is to be sent. Valid inputs are 1 or 2 for use in the CIM laboratory. The second parameter is the partition number where the program is to be downloaded. Valid inputs are 1, 2, 3, 4 or 5. The third parameter is the name of the program that is to be downloaded to the robot controller.

The robot number and partition are pointers to integers whereas the name of the AML/E program is a pointer to a character. The function starts by checking the accuracy of the input parameter against the values expected. It checks the robot number for values of 1 or 2 and the partition number for values 1 through 5. Any other value returns an error code to the calling function.

The first record of a compiled AML/E program is a N record which tells the controller the name of the application program and the partition into which the program

has to be loaded. A listing of the N record is given in Appendix G. The partition into which the program is loaded is the third character of the N record and its value has to be changed depending upon the value of the partition parameter. The variable send_char1 is set to the value of the chosen partition. The partition to which the program is downloaded is stored in a global variable, PARTT.

Next, the function opens the specified text file and returns an error code if the file does not exist. If it finds the file, it starts downloading it character by character using the primitive function SEND_COM(). While transmitting the N record, the function DOWNLOADROB() sends send_char1 as the third character in order for the controller to download to the right partition. The robot controller requires that each record must end with a carriage return/ line feed sequence. Thus, the function appends a Cr (Hex 0A) and Lf (Hex 0D) at the end of each line of the code it transmits. At the end of the file, it terminates the transmission by sending an end-of-file character. During the entire process of downloading, if an error occurs, the function returns an error code to the calling program. If the downloading process has been successful, DOWNLOADROB() returns an error code of 0 and exits. The four possible error codes are:

0 Downloaded program

1	Robot number error
2	Partition error
3	Could not open file

STOP_CYCLE(int *number)

The STOP_CYCLE() function is used for stop the execution of a program from a user specified partition. It has one parameter associated with it; the number of the robot which has to be stopped. Valid inputs are 1 and 2. The robot number parameter, number, is a pointer to an integer.

The function starts by checking for the accuracy of the input parameters against the values expected. In case there is an error, the function returns an error code to the calling program. The next block of code deals with the protocol for stopping the cycle. The function sends the robot controller an X identifier (hex 58) to indicate the start of an execute transaction using the SENDCOM() primitive function. The controller always responds with an initial Xoff (hex 13). If the controller is able to accept the record an Xon (hex 11) is sent following the Xoff. Both these characters are read by using the primitive function RECEIVECOM(). The STOP_CYCLE() function then sends the stop cycle operand "23" (hex 32, hex 33) to the controller

end of the record using SENDCOM(). The controller responds with an Ack (hex 06) to indicate that the record was received correctly, and that the command has completed. STOP_CYCLE() then sets the value of the global variable which is keeping track of the robot state, STATEROBOT, to 1 and returns to the calling program. In case of an error in the transactions between the robot controller and the STOP_CYCLE() function, the function returns an error code to the calling function and exits. The possible error codes are:

0	No error
1	Error in robot number

APPLICATION*()

The APPLICATION*() function is for choosing the partition specified by the user in the START_CYCLE() function. The function starts by checking the value of the global variable PARTT. PARTT is a global variable whose value is set to 1 in the DOWNLOAD() function if a program has been downloaded to that partition. Thus, APPLICATION*() checks for PRATT to be set and if it is not, APPLICATION*() returns an error code to START_CYCLE().

The next block of code deals with the protocol for activating the specified partition. The function sends the

robot controller an X identifier (hex 58) to indicate the start of an execute transaction using the SENDCOM() primitive function. The controller always responds with an initial Xoff (hex 13). If the controller is able to accept the record an Xon (hex 11) is sent following the Xoff. Both these characters are read by using the primitive function RECEIVECOM(). The APPLICATION*() function then sends the start cycle operand "3*" (hex 33, hex 3*) to the controller followed by the Cr (hex 0D) and Lf (hex 0A) to indicate the end of the record using SENDCOM(). The controller responds with an Ack (hex 06) to indicate that the record was received correctly, and that the command has completed. The "*" used in the above explanation may be replaced by the partition number (valid values are from 1 to 5) the user is activating by using the APPLICATION*() function eg. using APPLICATION3() would activate partition 3 of the robot controller. If the protocol has been successful, the function returns an error code of 0 and exits. The two possible error codes are:

0	No error
1	Partition empty

HALT_CYCLE(int *number)

The HALT_CYCLE() function is used for halting the execution of the robot and taking it back to the uninitialized state. It has one parameter associated with it; the number of the robot which has to be started. Valid inputs are 1 and 2. The robot number parameter, number, is a pointer to an integer.

The function starts by checking for the accuracy of the input parameters against the values expected. In case there is an error, the function returns an error code to the calling program. The next block of code deals with the protocol for stopping the cycle. The function sends the robot controller an X identifier (hex 58) to indicate the start of an execute transaction using the SENDCOM() primitive function. The controller always responds with an initial Xoff (hex 13). If the controller is able to accept the record an Xon (hex 11) is sent following the Xoff. Both these characters are read by using the primitive function RECEIVECOM(). The HALT_CYCLE() function then sends the stop cycle operand "23" (hex 32, hex 33) to the controller followed by the Cr (hex 0D) and Lf (hex 0A) to indicate the end of the record using SENDCOM(). The controller responds with an Ack (hex 06) to indicate that the record was received correctly, and that the command has completed.

HALT_CYCLE() then sets the value of the global variable which is keeping track of the robot state, STATEROBOT, to 0 and returns to the calling program. In case of an error in the transactions between the robot controller and the HALT_CYCLE() function, the function returns an error code to the calling function and exits. The possible error codes are:

0	No error
1	Error in robot number

MC_STATUS(int *number)

The MC_STATUS() function is used during the operation of the robot to obtain its current state and to identify the occurrence of any special error conditions. The function expects to be passed a pointer to the robot number whose state is being polled. In the present application, valid inputs are either 1 or 2 and any other value will return an error condition. MC_STATUS() returns the values of the errors in a structure to the calling program.

The MC_STATUS() function sends the controller an R-identifier (hex 53) to start a read transaction using the primitive function COMSEND(). The R tells the controller that the next thing sent will be an R record. The controller always responds with an initial Xoff (hex 13).

If the controller is able to accept the record an Xon (hex 11) is sent following the Xoff. If the controller is not ready additional Xoffs are sent about every 25 seconds until the controller is ready. These characters are read using the primitive function RECEIVECOM(). Next, the MC_STATUS() function sends the read controller status operand "01" (hex 30, hex 31) to the controller followed by the Cr (hex 0D) and Lf (hex 0A) which indicates the end of the record using the SENDCOM() function.

The controller responds with a D to indicate a data record eg. "D02044044"CrLf. The D identifier is followed by the byte count(02), the data(0440), the check sum(44) and the CrLf to indicate the end of the record. These characters are read using the primitive function RECEIVECOM() and stored in an array s[i]. The value of the data bytes varies depending on the error being reported by the robot controller. For example; in the data bytes above 04 shows a data error and 40 shows a point out of work space. The MC_STATUS() sends an Ack (hex 06) to tell the controller that the record was received correctly using primitive function SENDCOM(). The controller responds with an E record to show the end of the data. The record is as follows: an E to show this is an E record, a G to show the data was good, the CrLf shows the end of the record. These characters are read using the primitive function

RECEIVECOM(). The MC_STATUS() sends an Ack(hex 06) to tell the controller that the record was received correctly using the SENDCOM() function.

Next, the ASCII value of the data characters is taken from the array s[i], and is converted to its numerical value. Then the value of the first and second byte is computed and stored in bfirst and bsecond respectively. These values and the value of STATEROBOT are then assigned to the variables of a declared structure and returned to the calling program.

REJECT_STATUS(int *number)

The REJECT_STATUS() function is used during the operation of the robot to read the reject status of the robot controller. The controller gets into a reject status when it sends an end-of-transmission (EOT). The function expects to be passed a pointer to the robot number whose state is being polled. In the present application, valid inputs are either 1 or 2 and any other value will return an error condition.

The REJECT_STATUS() function sends the controller an R-identifier (hex 53) to start a read transaction using the primitive function COMSEND(). The R tells the controller

that the next thing sent will be an R record. The controller always responds with an initial Xoff (hex 13). If the controller is able to accept the record an Xon (hex 11) is sent following the Xoff. If the controller is not ready additional Xoffs are sent about every 25 seconds until the controller is ready. These characters are read using the primitive function RECEIVECOM(). Next, the REJECT_STATUS() function sends the read reject status operand "02" (hex 30, hex 32) to the controller followed by the Cr (hex 0D) and Lf (hex 0A) which indicates the end of the record using the SENDCOM() function.

The controller responds with a D to indicate a data record eg. "D014040"CrLf. The D identifier is followed by the byte count(02), the data(40), the check sum(40) and the CrLf to indicate the end of the record. These characters are read using the primitive function RECEIVECOM() and stored in an array s[i]. The value of the data byte varies depending on the error being reported by the robot controller. For example; in the data byte 40 shows a point out of work space. The MC_STATUS() sends an Ack (hex 06) to tell the controller that the record was received correctly using primitive function SENDCOM(). The controller responds with an E record to show the end of the data. The record is as follows: an E to show this is an E record, a G to show the data was good, the CrLf shows the end of the record.

These characters are read using the primitive function RECEIVECOM(). The MC_STATUS() sends an Ack(hex 06) to tell the controller that the record was received correctly using the SENDCOM() function.

Next, the ASCII value of the data characters is taken from the array s[i], and is converted to its numerical value. Then the value of the first byte is computed and stored in bfirst. The error associated with the first byte is then returned by comparing the values of bfirst with the expected values. This function returns the error codes associated with these error conditions.

RETURN_HOME()

The RETURN_HOME() function is used for returning the robot to the home position. The function sends the robot controller an X identifier (hex 58) to indicate the start of an execute transaction using the SENDCOM() primitive function. The controller always responds with an initial Xoff (hex 13). If the controller is able to accept the record an Xon (hex 11) is sent following the Xoff. Both these characters are read by using the primitive function RECEIVECOM(). The RETURN_HOME() function then sends the start cycle operand "11" (hex 31, hex 31) to the controller followed by the Cr (hex 0D) and Lf (hex 0A) to indicate the

end of the record using SENDCOM(). The controller returns home sending an Xoff every 25 seconds until home is reached; then it sends an Xon to indicate that the command is completed. The RETURN_HOME() function uses the primitive function RECEIVECOM() for reading in these values. If there is an error in the protocol, the function returns an error code. If not, the function exits and returns control to the calling program.

AUTOSTART()

The AUTOSTART() function is used in the initialization process of the robot. The function sends the robot controller an X identifier (hex 58) to indicate the start of an execute transaction using the SENDCOM() primitive function. The controller always responds with an initial Xoff (hex 13). If the controller is able to accept the record an Xon (hex 11) is sent following the Xoff. Both these characters are read by using the primitive function RECEIVECOM(). The AUTOSTART() function then sends the auto operand "20" (hex 32, hex 30) to the controller followed by the Cr (hex 0D) and Lf (hex 0A) to indicate the end of the record using SENDCOM(). The controller responds with an Ack (hex 06) to indicate that the record was received correctly, and the command has completed. AUTOSTART() reads in this value by using the primitive function RECEIVECOM(). Just

before exiting the AUTOSTART() function, the value of the global variable STATEROBOT is set equal to 1. After this the function exits and returns control to the calling program.

The functions described above have made use of some primitive functions for basic communication and control. A detailed description of these primitive functions has been given in Chapter 7. These communication primitives aid in communication between devices, help in configuration of the ports, and help identify the status of the communication port.

9. MENU DRIVER FUNCTIONS

The usage of the functions created in the milling and robot function libraries has been demonstrated by a menu driven program. The menus let the user interact with a particular machine function library and control the machine at various levels. The menu driven program helps familiarize the user with the library functions and the parameters associated with them. Additionally, the menu driven programs provide a framework for basic control of the various machines by operator input.

The menu has three levels. The top most level is the MAINMENU() where the user selects the machine to be controlled. The next level has the machine menus. For example NC1MENU() is a function which gives the user all the options associated with the control of CNC machine #1. The user may INITIALIZE(), DOWNLOAD(), HALT() or get the STATUS() of the machine. There are two additional machine menu functions for CNC machine #2 and the robot. The third level menus are in reality dialog menus for user inputted parameters for the functions chosen in level 2.

The main menu is presented to the user when the program is started. The main menu screen is illustrated in Figure 5. The user can select the machine to be controlled. The choice is between the Robot, CNC machine #1 or CNC machine #2. Selecting any of these options brings up the menus at the lower levels. The user may exit the system by choosing option 4.

The next level has the machine menus. There are three menus on this level associated with the Robot, CNC machine #1 and CNC machine #2 respectively. The robot menu screen is shown in Figure 6. This gives the user the options for controlling the robot. The user can initialize, download to, poll for status, or stop the robot. Selecting option 7 makes the user exit this screen and go up one level, to the main menu.

The CNC machine #1 and #2 also have menus associated with their control at the level discussed above. Figure 7 illustrates the CNC machine #1 screen. The options presented give the user the options in controlling the CNC machine. CNC machine #2 has a similar menu. In reality, both the CNC machine menus call the same library functions with different parameters. The user may exit the menu by choosing option 6 which takes him back to the main menu.

SELECT FROM MENU OPTIONS

1. ROBOT

2. NC MACHINE #1

3. NC MACHINE #2

4. EXIT

Enter number of option chosen ..

Figure 5: Main menu screen

SELECT FROM ROBOT MENU OPTIONS

1. INITIALIZE()

2. DOWNLOAD()

3. GO()

4. STATUS()

5. STOP()

6. HALT()

7. EXIT

Enter number for option chosen

Figure 6: Robot menu screen

SELECT FROM CNC #1 MENU OPTIONS

1. INITIALIZE()

2. DOWNLOAD()

3. GO()

4. STATUS()

5. HALT()

6. EXIT

Enter number of option chosen ..

Figure 7: CNC machine #1 screen

ENTER THE PARAMETERS FOR INITIALISATION

Enter the CNC machine number:_1

Enter initialising program name:_mpd

Enter the forcing function:_4

ENTER PARAMETERS FOR THE DOWNLOADING

Enter the NC machine number:_1

Enter program name:_cut012

Figure 8: Input menus of CNC machine #1 with responses

ENTER THE PARAMETERS FOR THE DOWNLOAD COMMAND

Enter the robot number:_1

Enter the partition number:_4

Enter the program name:_nod1

Figure 9: Robot DOWNLOAD() inputs with example responses

The next level handles the menus for inputting the parameter of the called functions. Figure 8 and 9 show some input menus with example responses. After the user has specified the parameter, the program processes the command and executes it on the chosen machine. The return values of the called library functions are also processed at level 3 and the messages are output on the screen. These return values may indicate status and operational or machine errors.

10. CONCLUSIONS AND RECCOMENDATIONS

The interface between the robot, CNC machines and the cell controller was successfully developed as part of this project. The library of functions created for the Dyna CNC machine and the IBM 7545/7547 robots will aid the future user to create software for the work cell. By using the library functions, a user will be insulated from the low level functioning of the machines and need only be concerned about the operation of these functions.

Specifically created as part of this research were functions to control the operations of a Dyna 2400 CNC machine. This included functions to initialize the machine, download and execution of Dyna programs, monitor machine status, and perform an orderly shutdown. Similar functions were created for interface to the IBM 7545/7547 robot controllers. Additionally a set of menu driven functions were created to exercise the control functions and to give the system user keyboard control over the operation of the sub-systems.

Future work should be directed in modifying the hardware and software to give more capability to the work cell controller. In the machining cell, if three machines are to be controlled on a real time basis, there is a need for three communication ports. As the current version of MS-DOS supports only COM1 and COM2, future development would require the use of a serial interface board which provides multi-channel interface capability. The existing software could then be modified to access all three ports.

Proper functioning of each work cell requires digital input/output access to the control computer. This is necessary to interface various work cell sensors and actuators. Presently the plan requires interfacing some of these input/output signals directly to the work cell controllers while others will interface to a programmable controller (which will also be interfaced to the work cell controller). The programmable controller will permit some work cell mechanisms to operate in parallel with the work cell controller serving primarily as a process initiator and status monitor.

A special function was added to the milling machine library to interface with the digital input and output lines of the Dyna milling machine. This would permit the work cell controller and the milling machine controller to have

better control of a downloaded program. The GO() function was written to coordinate program execution via digital input/output. Since digital input/output control was not implemented in this research the function is only included for completeness. Further work in this area is necessary for improved program control.

The interfacing software created as part of this project can be further enhanced by including these additional routines. The DOWNLOAD() function in the Dyna CNC machine can be further improved by adding the capability of validating the text file that is being downloaded. The DYNA controller requires the program being downloaded to be in a particular format but it cannot detect any deviation. Thus the downloading of a incorrect file results in the CNC controller going into an error condition. This corruption of the functioning of the CNC can be avoided if the function has the additional capability of validation of the text file which is being downloaded.

In the robot function library, the DIDO_STATUS() function has been written to provide the user the status of the DI/DO lines. This could not be validated because of the lack of a DI/DO port on the work cell controller. Also, the STATUS() function of the robot returns the reason for the error but does not reset the error. Future modification

should be directed in modifying the DIDO_STATUS() function and creating a ERRESET() function for resetting the error.

11. REFERENCES

1) Arai. Y., S. Hata, T. Imakubo and K. Kikuchi, "Production Control Systems of Microcomputer Hierarchical Structures for FMS," Proceedings of the 1st International Conference on FMS, Brighton, UK, October 20-22, pp. 259-268, 1982.

3) Bertrand, J. W. M., "A Hierarchical Approach to Structuring the Production Control in Multi-Product Multi-Phase Production Systems," Modeling Production Modeling Systems, North-Holland, pp. 57-76, 1984.

4) Duffie, N. A. and R. S. Piper, " Non-hierarchical control of a Flexible Manufacturing Cell," Robotics and Computer Integrated Manufacturing, Vol. 3, No. 2, pp. 175-179, 1987.

5) Dutton, D., "The design and implementation of a cell controller," M. S. Thesis, Georgia Institute of Technology, December 1987.

6) Flexible Manufacturing Systems Handbook, The Charles Stark Draper Laboratory. Noyes Publication, Park Ridge, NJ, pp. 28-36, 152-164, 1984.

7) Jones, A. T. and C. R. McLean, "A Proposed Hierarchical Control Model for Automated Manufacturing Systems," Journal of Manufacturing Systems, Vol. 5, No. 1, pp. 15

-25, November 1987.

8) Palframan, D., "FMS- Too Much, Too Soon," Manufacturing Engineering, March, 1987.

9) Reference Model of Production Systems, Version 1.0, Digital Equipment Corporation and Nederlandse Philips Bedrijven BV, 1987.

10) Richardson, D., "Implementing MAP for Factory Control," Manufacturing Engineering, January, 1988.

11) Seliger, G. and B. Viehweger, "Concepts of Flexible Automated Manufacturing," Journal of Manufacturing Systems, Vol. 5, No. 3, pp. 171-178, April 1987.

12) Tanenbaum, A. S., Computer Networks, Prentice- Hall Inc., Englewood Cliffs, NJ, 1981.

13) DM 2400/ 2200 Programming Manual, DYNA Electronics Inc., 2346 Walsh Avenue, Santa Clara, CA 95051, 1984.

14) IBM 7545 Manufacturing System Hardware Library, IBM Corp., AMS Information Development, P.O. Box 1328, Boca Raton, FL 33432.

APPENDIX A: PROGRAM LISTING

```
/*********************************************************/
/* THE FIRST SECTION OF THE CODE CONTAINS THE DYNA      */
/* FUNCTION LIBRARY AND THE MENU FUNCTIONS              */
/*********************************************************/


#include <stdio.h>
#include <conio.h>
#include <bios.h>
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>


static char eutto[] = {'e','n','d'};
static int COUNT ;
static int STATE ;
unsigned comdata;
unsigned com_status;
unsigned com_error;
unsigned com_data;
unsigned com_port= 0;
unsigned data_bits;
unsigned stop_bits;
unsigned parity;
unsigned baud_rate;
unsigned char send_char, goat;
void mainmenu(), robotmenu();
void nc1menu(),nc2menu(),cls(),goto_xy();
void initncmenu(),downlncmenu();
void goncmenu(),haltncmenu(),statusncmenu();
void draw_border();
int initialize(int *,char *,int *),download(int *, char *);
int endload(int *),status(int *);


void get_com_status (com_port, com_status)              /*
COM status routine        */
```

```
  unsigned *com_status;
  {
    *com_status = _bios_serialcom (_COM_STATUS, com_port,
0);
  }


 void mainmenu()
    {
     int i,rtn;
     short command;
     for (i =0;i<25; i++)
     printf("\n");
     do
     {
     cls();
     draw_border(7,20,19,59);
     goto_xy(9,25);
     printf(" SELECT FROM MENU OPTIONS");
     goto_xy(11,30);
     printf ("1. ROBOT");
     goto_xy(12,30);
     printf("2. NC MACHINE #1");
     goto_xy(13,30);
     printf("3. NC MACHINE #2");
     goto_xy(14,30);
     printf("4. EXIT");
     goto_xy(17,22);
     printf(" Enter number for Option chosen ..");
     goto_xy(17,56);
     rtn = scanf ("%3d", &command );
     if (rtn == 0)
        command = 0;
     else if (command > 4)
        command = 5;
     switch (command)
            {
            case 1:
            robotmenu();
            break;
            case 2:
            nc1menu();
            break;
            case 3:
            nc2menu();
            break;
            case 4:
            break;
            default:

            printf ("Unknown Command: %2d\7",command);
```

```
            break;
            }
    }
   while ( command != 4);
   for (i =0;i<25; i++)
   printf("\n");
   }

void robotmenu()
   {
    int rtn;
    short command;
    do
    {
    cls();
    draw_border(7,19,20,60);
    goto_xy(9,22);
    printf(" SELECT FROM ROBOT MENU OPTIONS");
    goto_xy(11,30);
    printf ("1. INITIALIZE()");
    goto_xy(12,30);
    printf("2. DOWNLOAD()");
    goto_xy(13,30);
    printf("3. GO()");
    goto_xy(14,30);
    printf("4. STATUS()");
    goto_xy(15,30);
    printf("5. HALT()");
    goto_xy(16,30);
    printf("6. EXIT");
    goto_xy(18,22);
    printf(" Enter number for Option chosen ..");
    goto_xy(18,55);
    goto_xy(18,57);
    rtn = scanf ("%hd", &command );

    if (rtn == 0)
       command = 0;
    switch (command)
           {
           case 1:
           initrobotmenu();
           break;
           case 2:
           downlrobotmenu();
           break;
           case 3:
            gorobotmenu();
           break;
           case 4:
            statusrobotmenu();
```

```
            break;
            case 5:
              haltrobotmenu();
            break;
            case 6:
            break;
            default:
            printf ("Unknown Command: %d\7",command);
            break;
            }
     }
    while ( command != 6);
    }


void nc1menu()
    {
     short command;
     do
     {
     cls();
     draw_border(7,19,20,60);
     goto_xy(9,22);
     printf(" SELECT FROM CNC # 1 MENU OPTIONS");
     goto_xy(11,30);
     printf ("1. INITIALIZE()");
     goto_xy(12,30);
     printf("2. DOWNLOAD()");
     goto_xy(13,30);
     printf("3. GO()");
     goto_xy(14,30);
     printf("4. STATUS()");
     goto_xy(15,30);
     printf("5. HALT()");
     goto_xy(16,30);
     printf("6. EXIT");
     goto_xy(18,22);
     printf(" Enter number for Option chosen _");
     goto_xy(19,56);
     scanf ("%hd", &command );

     switch (command)
            {
            case 1:
            initncmenu();
            break;
            case 2:
            downlncmenu();
            break;
            case 3:
            goncmenu();
```

```
            break;
            case 4:
            statusncmenu();
            break;
            case 5:
            haltncmenu();
            break;
            case 6:
            break;
            default:
            printf ("Unknown Command: %d\7",command);
            break;
            }
     }
    while ( command != 6);
    }


void nc2menu()
    {
     int rtn;
     short command;
     do
     {
     cls();
     draw_border(7,19,20,60);
     goto_xy(9,22);
     printf(" SELECT FROM CNC # 2 MENU OPTIONS");
     goto_xy(11,30);
     printf ("1. INITIALIZE()");
     goto_xy(12,30);
     printf("2. DOWNLOAD()");
     goto_xy(13,30);
     printf("3. GO()");
     goto_xy(14,30);
     printf("4. STATUS()");
     goto_xy(15,30);
     printf("5. HALT()");
     goto_xy(16,30);
     printf("6. EXIT");
     goto_xy(18,22);
     printf(" Enter number for Option chosen ..");
     goto_xy(18,57);
     rtn = scanf ("%hd", &command );

     if (rtn == 0)
        command = 0;
     switch (command)
            {
            case 1:
            initncmenu();
```

```
            break;
            case 2:
            downlncmenu();
            break;
            case 3:
            goncmenu();
            break;
            case 4:
            statusncmenu();
            break;
            case 5:
            haltncmenu();
            break;
            case 6:
            break;
            default:
            printf ("Unknown Command: %d\7\n",command);
            break;
            }
      }
    while ( command != 6);
    }


 void initncmenu()

    {
     int number, ffun, rtn;
     char name[20];
     cls();
     draw_border(8,18,18,66);
     goto_xy(10,20);
     printf ("ENTER THE PARAMETERS FOR THE INITIALISATION
");
     goto_xy(12,25);
     printf ("Enter the NC machine number:__");
     goto_xy(12,55);
     scanf ("%d", &number);
     goto_xy(14,25);
     printf ("Enter initialising program name:__");
     goto_xy(14,60);
     scanf ("%s",name);
     goto_xy(16,25);
     printf ("Enter the forcing function__");
     goto_xy(16,55);
     scanf ("%d", &ffun );
     rtn = initialize (&number,name, &ffun);
     draw_border(21,24,23,54);
     switch (rtn)
     {
     case 0:
```

```
        goto_xy(22,25);
        printf (" Initializing was correct ");
        while (!kbhit());
        break;
      case 1:
        goto_xy(22,25);
        printf (" Machine already initialized\7");
        while (!kbhit());
        break;
      case 2:
        goto_xy(22,25);
        printf (" Machine code error\7");
        while (!kbhit());
        break;
      case 3:
        goto_xy(22,25);
        printf (" Forcing function error\7");
        while (!kbhit());
        break;
      case 4:
        goto_xy(22,25);
        printf (" Machine not ready\7");
        while (!kbhit());
        break;
      case 5:
        goto_xy(22,25);
        printf (" Initializing file error\7");
        while (!kbhit());
        break;
      default:
        break;
       }
}


 void downlncmenu()

     {
      int number,rtn;
      char name[20];
      cls();
      draw_border(8,18,16,62);
      goto_xy(10,20);
      printf ("ENTER THE PARAMETERS FOR THE DOWNLOADING ");
      goto_xy(12,25);
      printf ("Enter the NC machine number:__");
      goto_xy(12,55);
      scanf ("%d", &number);
      goto_xy(14,25);
      printf ("Enter program name:__");
      goto_xy(14,48);
```

```
scanf ("%s",name);
rtn = download (&number,name);
draw_border(21,24,23,54);
switch (rtn)
{
case 0:
  goto_xy(22,25);
  printf (" Download was correct ");
  while ( !kbhit());
  break;
case 1:
  goto_xy(22,25);
  printf (" Machine not initialized \7");
  while ( !kbhit());
  break;
case 2:
  goto_xy(22,25);
  switch (STATE)
     {
     case 2:
      printf (" Absolute zero not set");
      while ( !kbhit());
      break;
     case 3:
      printf (" Machine executing program");
      while ( !kbhit());
      break;
     }
  break;
case 3:
  goto_xy(22,25);
  printf (" Machine code error \7");
  while ( !kbhit());
  break;
case 4:
  goto_xy(22,25);
  printf (" File not found\7");
  while ( !kbhit());
  break;
case 5:
  goto_xy(22,25);
  printf (" Not valid text file\7");
  while ( !kbhit());
  break;
default:
  break;
}

}
```

```
void goncmenu()

    {
     int number;
     cls();
     printf ("Enter the parameters needed for the selected
function\n\n\n");
     printf ("Enter the NC machine number:______\n\n");
     scanf ("%d", &number);
    }

void statusncmenu()

    {
     int number, rtn;
     cls();
     draw_border(8,18,14,57);
     goto_xy(10,20);
     printf ("ENTER THE PARAMETERS FOR THE STATUS ");
     goto_xy(12,25);
     printf ("Enter the NC machine number:__");
     goto_xy(12,55);
     scanf ("%d",&number);
     rtn = status (&number);
     draw_border(18,23,20,52);
     switch (rtn)
     {
     case 0:
       goto_xy(19,25);
       printf (" Uninitialized, not ready ");
       while (!kbhit());
       break;
     case 1:
       goto_xy(19,25);
       printf (" Uninitialized, Ready");
       while (!kbhit());
       break;
     case 2:
       goto_xy(19,25);
       printf (" Initialized");
       while  (!kbhit());
       break;
     case 3:
       goto_xy(19,25);
       printf (" Downloaded");
       while (!kbhit());
       break;
     case 4:
       goto_xy(19,25);
       printf (" Active, not ready");
       while  (!kbhit());
```

```
      break;
    case 7:
      goto_xy(19,25);
      printf("Incorrect machine code");
      while (!kbhit());
      break;
    default:
      break;
     }
}


void haltncmenu()

    {
     int number, rtn;
     cls();
     draw_border(8,18,14,57);
     goto_xy(10,20);
     printf ("ENTER THE PARAMETERS FOR THE HALT ");
     goto_xy(12,25);
     printf ("Enter the NC machine number:__");
     goto_xy(12,55);
     scanf ("%d",&number);
     rtn = endload (&number);
     draw_border(18,23,20,52);
     switch(rtn)
      {
     case 0:
          goto_xy(19,25);
          printf (" End done properly");
          while (!kbhit());
          break;
     case 1:
          goto_xy(19,25);
          printf (" Machine code error");
          while (!kbhit());
          break;
     case 2:
          goto_xy(19,25);
          printf (" Could not open file ");
          while (!kbhit());
          break;
     default:
          break;
      }
    }

int initialize ( ncnumber, name, ffunction )
int *ncnumber;
```

```
int *ffunction;
char *name;
    {
     FILE *note;
     int i;
     extern int STATE, COUNT;

     if (*ncnumber != ( 1 || 2)) {
     return (2);                        /* machine code error */
     }
     if (*ffunction != ( 2 || 1)) {
     return (3);
     }
     if ( *ffunction == 1 && COUNT == 1 ) {
     return (1);
     }
       if (( *ffunction == 1 && COUNT == 0) ||
(*ffunction==2 && COUNT ==1))
      {
         if ((note = fopen(name,"r")) != NULL)
          {
          while ((send_char=getc(note)) && (feof (note) ==
0))
          {
                send_com (com_port, &send_char, &com_error);
                    if (send_char == 10 )

                   {   goat = 13;
                    send_com (com_port, &goat, &com_error);
                   }
          }
          if (feof (note))
                {
               send_char = 26;
               send_com (com_port, &send_char, &com_error);
                 }
                fclose (note);
                if ( *ffunction == 1 && COUNT == 0) {
               COUNT =1;
               }
                STATE = 2;
                return (0);
          }
          else
               return (5);
         }
     }


int download (ncnumber, name)  /* downloading of program */
int *ncnumber;
```

```
char *name;
{
 FILE *notes;
 char cool;
   if ( *ncnumber != ( 1 || 2 ))
      {
     return(3);
      }
   switch ( STATE )
     {
     case 2:
     break;
     case 3:
     break;
     default:
     return (1);
     }
   get_com_status(com_port,&com_status);
      if ((com_status & 16) == 0)
        return(2);

   if ((notes = fopen(name,"r")) != NULL)
     {
      while ((send_char=getc(notes)) && (feof (notes) == 0))
          {
            send_com (com_port, &send_char, &com_error);
            if (send_char == 10 )
                {   goat = 13;
                 send_com (com_port, &goat, &com_error);
                }
           }
      if (feof (notes))
           {
            send_char = 26;
            send_com (com_port, &send_char, &com_error);
           }
      fclose (notes);
      STATE = 3;
      return (0);
     }
     else
     return (4);
  }


int endload (ncnumber)
int *ncnumber;
{
  FILE *note;
  extern char eutto[3];
```

```
  switch ( *ncnumber)
    {
     case 1:
    break;
     case 2:
    break;
     default:
    return (1);
    }
  if ((note = fopen(eutto,"r")) != NULL)
          {

          while ((send_char=getc(note)) && (feof (note) ==
0))

          {
                send_com (com_port, &send_char, &com_error);
                    if (send_char == 10 )

                   {   goat = 13;
                    send_com (com_port, &goat, &com_error);

                    }
          }


          if (feof (note))

                 {
               send_char = 26;
               send_com (com_port, &send_char, &com_error);
                 }

          fclose (note);
          STATE =0;
          return (0);
           }
          else
               return (2);
     }


int status(mcnumber)
int *mcnumber;
   {
     int i;
     if ( *mcnumber != (1 || 2))
       return (7);
     draw_border(21,23,23,57);
     goto_xy(22,25);
```

```
     get_com_status(com_port, &com_status);
     if ((com_status & 16 ) == 0)
        {
         switch (STATE)
          {
          case 2:
           printf (" Absolute zero not set ");
           break;
          case 3:
           printf (" Machine executing program");
           break;
          default:
           break;
          }
        }
     else {
           printf (" Machine not active");
          }
     i =STATE;
     return (i);
 }

void init_com (com_port, com_data, com_error)     /* COM
initialization routine    */
unsigned *com_error;
  {
    if (com_port > 1)        /* Check for valid COM port  */
                         /*   only 1 COM     */
           /*   port is installed, put 0 instead of 1 */
      {
      printf ("Invalid COM port %u\n", com_port + 1);
      *com_error = 0xFFFF;   /* Set error flag if port
number out of range    */
      }
    else
      {
      *com_error = _bios_serialcom (_COM_INIT, com_port,
com_data);
      }
  }

send_com (com_port, com_char, com_error)       /* COM send
routine            */
  unsigned char *com_char;
  unsigned *com_error;
  {
    *com_error = _bios_serialcom (_COM_SEND, com_port,
*com_char) & 0xFF00;
   }

void receive_com (com_port, com_char, com_error)    /* COM
```

```
receive routine          */
  unsigned char *com_char;
  unsigned *com_error;
  {
    while ((com_status & 0x2000) < 0x2000)   /* Check
Transmission    */
      get_com_status (com_port, &com_status);       /*
Holding Register      */
    *com_char  = 0;
    *com_error = 0;
                    /* Repeat until a character was received
or   */
                    /* an error occurs.  If time-out should
be    */
                    /* enabled change 0x1E00 to 0x9E00
*/
    while (((*com_error & 0x1E00) < 0x0200) && (*com_char <
0x01))
      {
      *com_error = _bios_serialcom(_COM_RECEIVE,com_port,0);
                    /* Character is returned in low byte
*/
      *com_char  = (unsigned char) (*com_error & 0x00FF);
                    /* Error is returned in high    byte
*/
      *com_error = *com_error & 0xFF00;

      if (kbhit())
       break;
      }
  }

 void cls()
  {
   union REGS r;
   r.h.ah=6;
   r.h.al=0;
   r.h.ch=0;
   r.h.cl=0;
   r.h.dh=24;
   r.h.dl=79;
   r.h.bh=7;
   int86(0x10, &r, &r);
   }

 void goto_xy(x,y)
 int x,y;
 {
   union REGS r;

   r.h.ah=2;
```

```
   r.h.dl=y;
   r.h.dh=x;
   r.h.bh=0;
   int86(0x10, &r, &r);
 }

void draw_border(startx, starty, endx,endy)
  int startx, starty,endx,endy;
  {
  register int i;
  for (i= startx+1; i<endx; i++) {
  goto_xy(i,starty);
  putchar(179);
  goto_xy(i, endy);
  putchar(179);
  }

  for(i=starty+1;i<endy; i++) {
  goto_xy(startx, i);
  putchar(196);
  goto_xy(endx, i);
  putchar(196);
  }

  goto_xy(startx, starty); putchar(218);
  goto_xy(startx, endy); putchar(191);
  goto_xy(endx,starty); putchar(192);
  goto_xy(endx,endy); putchar(217);
  }

main ()

  {
  com_port  = 0;                  /* valid com_port  are:  0
(for COM1:)          */
                     /*                          1 (for COM2:)
*/
                     /* could be higher if more COM ports
present  */
  data_bits = _COM_CHR8;          /* valid data_bits are:
_COM_CHAR7             */
                     /*                          _COM_CHAR8
*/
  stop_bits = _COM_STOP1;         /* valid stop_bits are:
_COM_STOP1             */
                     /*                          _COM_STOP2
*/
  parity    = _COM_NOPARITY;      /* valid parity    are:
_COM_NOPARITY          */
                     /*                          _COM_EVENPARITY
*/
```

```
                    /*                          _COM_ODDPARITY
*/
  baud_rate = _COM_2400;            /* valid baud_rate are:
_COM_110            */
                    /*                          _COM_150
*/
                    /*                          _COM_300
*/
                    /*                          _COM_600
*/
                    /*                          _COM_1200
*/
                    /*                          _COM_2400
*/
                    /*                          _COM_4800
*/
                    /*                          _COM_9600
*/

                    /* Calculate initialization data by
ORing     */
  com_data  = data_bits | stop_bits | parity | baud_rate;

                    /* initialize COM error to zero (no
error)    */
                    /* Meaning of errors and status is shown
on   */
                    /* pages 144 to 145 of Microsoft C Run-
Time   */
                    /* Library Reference Manual
*/
  com_error = 0;
  comdata  = _COM_CHR7 | _COM_STOP2 | _COM_EVENPARITY |
_COM_4800;


                    /* initialize COM port with parameters
*/
/*  init_com (com_port, com_data, &com_error);*/
  printf("***********  %4X\n",initcom(comdata));
  getch();

/* check for COM status and print it          */
  get_com_status (com_port, &com_status);
  cls();
  mainmenu();
  get_com_status (com_port, &com_status);


  /*puts("Press any key to continue.\n");
  getch();
```

```
     receive character from COM port and print
  receive_com (com_port, &receive_char, &com_error);
  printf ("Received character %c from COM%lu\n",
receive_char, com_port + 1); */
  }
```

```
/*********************************************/
/*   THIS SECTION CONTAINS THE ROBOT MENU    */
/*             FUNCTIONS                     */
/*                                           */
/*********************************************/



#include <bios.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

void initrobotmenu();
void gorobotmenu();
void downlrobotmenu();
void haltrobotmenu();

 unsigned com_port;
 unsigned char send_char;
 unsigned com_error=0;
 unsigned char receive_char;



 void initrobotmenu()

    {
     int number;
     cls();
     draw_border(8,18,14,59);
     goto_xy(10,20);
     printf ("ENTER THE PARAMETERS FOR INITIALISING ");
     goto_xy(12,25);
     printf ("Enter the ROBOT number:__");
     goto_xy(12,55);
     scanf ("%hd",&number);
     draw_border(18,23,20,52);
     return_home();
     autostart ();
    }


 void gorobotmenu()

    {
     int number, part, rtn;
     cls();
     draw_border(8,18,16,66);
```

```
     goto_xy(10,20);
     printf ("ENTER THE PARAMETERS FOR THE GO COMMAND");
     goto_xy(12,25);
     printf ("Enter the Robot number:__");
     goto_xy(12,55);
     scanf ("%d", &number);
     goto_xy(14,25);
     printf ("Enter the partition number:__");
     goto_xy(14,60);
     scanf ("%d",&part);
     rtn = start_cycle (&number, &part);
     draw_border(21,24,23,54);
     switch (rtn)
     {
     case 0:
       goto_xy(22,25);
       printf (" Started Cycle ");
       while (!kbhit());
       break;
     case 1:
       goto_xy(22,25);
       printf (" Robot number error\7");
       while (!kbhit());
       break;
     case 2:
       goto_xy(22,25);
       printf (" Starting function error\7");
       while (!kbhit());
       break;
     case 4:
       goto_xy(22,25);
       printf("Partition empty");
       while ( !kbhit());
       break;
     default:
       break;
      }
}


 void downlrobotmenu()

    {
     int number, part, rtn;
     char name[20];
     cls();
     draw_border(8,18,18,66);
     goto_xy(10,20);
     printf ("ENTER THE PARAMETERS FOR THE DOWNLOAD
COMMAND");
     goto_xy(12,25);
```

```
     printf ("Enter the Robot number:__");
     goto_xy(12,55);
     scanf ("%d", &number);
     goto_xy(14,25);
     printf ("Enter the partition number:__");
     goto_xy(14,60);
     scanf ("%d",&part);
     goto_xy(16,25);
     printf ("Enter the program name:---");
     goto_xy(16,51);
     scanf ("%s",name);
     rtn = downloadrob (&number, &part,name);
     draw_border(21,24,23,54);
     switch (rtn)
     {
     case 0:
       goto_xy(22,25);
       printf (" Downloaded program ");
       while (!kbhit());
       break;
     case 1:
       goto_xy(22,25);
       printf (" Robot number error\7");
       while (!kbhit());
       break;
     case 2:
       goto_xy(22,25);
       printf (" Partition error\7");
       while (!kbhit());
       break;
     case 3:
       goto_xy(22,25);
       printf ("Could not open file\7");
       break;
     default:
       break;
      }
}


 void haltrobotmenu()

    {
     int number;
     cls();
     draw_border(8,18,14,59);
     goto_xy(10,20);
     printf ("ENTER THE PARAMETERS FOR HALTING ");
     goto_xy(12,25);
     printf ("Enter the ROBOT number:__");
     goto_xy(12,55);
```

```
 scanf ("%hd",&number);
 draw_border(18,23,20,52);
 stop_cycle(&number);
/* flushpart(); */
}
```

```
/*******************************************************/

/*  THIS SECTION CONTAINS THE ROBOT LIBRARY FUNCTIONS  */

/*******************************************************/


#include <bios.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>


  unsigned com_status;
  unsigned comdata;
  comerror =0;
  data_bits = _COM_CHR7;        /* valid data_bits are:
_COM_CHAR7                    */
                             /*                        _COM_CHAR8
*/
  stop_bits = _COM_STOP2;       /* valid stop_bits are:
_COM_STOP1                    */
                             /*                        _COM_STOP2
*/
  parity    = _COM_EVENPARITY;    /* valid parity    are:
_COM_NOPARITY                 */
                             /*                        _COM_EVENPARITY
*/
                             /*                        _COM_ODDPARITY
*/
  baud_rate = _COM_4800;         /* valid baud_rate are:
_COM_110                      */
                             /*                        _COM_150
*/
                             /*                        _COM_300
*/
                             /*                        _COM_600
*/
                             /*                        _COM_1200
*/
                             /*                        _COM_2400
*/
                             /*                        _COM_4800
*/
                             /*                        _COM_9600
*/

                             /* Calculate initialization data by
ORing     */
```

```
                         /* initialize COM error to zero (no
error)    */
                         /* Meaning of errors and status is shown
on   */
                         /* pages 144 to 145 of Microsoft C Run-
Time   */
                         /* Library Reference Manual
*/

   static int STATEROBOT;
   static int PARTT1;
   static int PARTT2;
   static int PARTT3;
   static int PARTT4;
   static int PARTT5;

#define WHICHCOMPORT 0

typedef struct mc_stat
     {
     int first;
     int second;
     int staterobot;
     }MC_STAT;

typedef struct rej_stat
     {
     int rejstat;
     }REJ_STAT;
   struct input_1
     {
       unsigned I_1:1;                 /* define the unit
fields */
       unsigned I_2:1;                 /* of the variables
input_1, */
       unsigned I_3:1;                 /* input_2, output_1,
output_2 */
       unsigned I_4:1;
       unsigned I_5:1;
       unsigned I_6:1;
       unsigned I_7:1;
       unsigned I_8:1;
     } ;
   struct  input_2
     {
       unsigned I_9:1;
       unsigned I_10:1;
       unsigned I_11:1;
       unsigned I_12:1;
       unsigned I_13:1;
```

```
        unsigned I_14:1;
        unsigned I_15:1;
        unsigned I_16:1;
      } ;
    struct  output_1
      {
        unsigned O_1:1;
        unsigned O_2:1;
        unsigned O_3:1;
        unsigned O_4:1;
        unsigned O_5:1;
        unsigned O_6:1;
        unsigned O_7:1;
        unsigned O_8:1;
      } ;
    struct  output_2
      {
        unsigned O_9:1;
        unsigned O_10:1;
        unsigned O_11:1;
        unsigned O_12:1;
        unsigned O_13:1;
        unsigned O_14:1;
        unsigned O_15:1;
        unsigned O_16:1;
      } ;


/*--------------------------------------------- function
prototypes ------*/
unsigned sendcom ();
void statusrobotmenu();
void ack();
void flushpart();
int stop_cycle(int *);
int start_cycle(int *,int *);
void return_home();
void autostart();
void di_do_status();
unsigned getcomstatus();
int downloadrob(int *,int *, char *);
MC_STAT mc_status(int *);
REJ_STAT reject_status();
int application1();
int application2();
int application3();
int application4();
int application5();
unsigned initcom ();
static char ptr[] ={'h','a','l','t','r','o','b','o','t'};
```

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

void statusrobotmenu()

   {
    int number;
    MC_STAT rtn;
    REJ_STAT gotit;
    cls();
    draw_border(6,18,12,59);
    goto_xy(8,20);
    printf ("ENTER THE PARAMETERS FOR STATUS ");
    goto_xy(10,25);
    printf ("Enter the ROBOT number:__");
    goto_xy(10,55);
    scanf ("%hd",&number);
    draw_border(14,56,24,78);
    goto_xy(15,58);
    if ( PARTT1 ==1)
    {
    printf ("Partition 1: F");
    }
    else
    {
    printf ("Partition 1: E");
    }
    goto_xy(17,58);
    if ( PARTT2 ==1)
    {
    printf ("Partition 2: F");
    }
    else
    {
    printf ("Partition 2: E");
    }
    goto_xy(19,58);
    if ( PARTT3 ==1)
    {
    printf ("Partition 3: F");
    }
    else
    {
    printf ("Partition 3: E");
    }
    goto_xy(21,58);
    if ( PARTT4 ==1)
    {
    printf ("Partition 4: F");
    }
    else
```

```
{
printf ("Partition 4: E");
}
goto_xy(23,58);
if ( PARTT5 ==1)
{
printf ("Partition 5: F");
}
else
{
printf ("Partition 5: E");
}
 rtn =mc_status( &number );
 gotit = reject_status();

draw_border(22,24,24,54);

switch (rtn.staterobot)
{
case 0:
  goto_xy(23,25);
  printf (" Uninitialized ");
  break;
case 1:
  goto_xy(23,25);
  printf ("Initialized; Download program");
  break;
case 2:
  goto_xy(23,25);
  printf (" In operation");
  break;
case 3:
  goto_xy(23,25);
  printf ("Incorrect M\C number\7");
  break;
default:
  break;
  }
draw_border(16,24,18,54);
goto_xy(17,25);

switch(rtn.first)
{
case 128:
      printf(" Servo error");
      break;
case 64:
      printf("Power failure");
      break;
case 32:
      printf("Overrun");
```

```
        break;
 case 16:
        printf("Over time");
        break;
 case 8:
        printf("Transmission error");
        break;
 case 0:
        printf("No machine status error");
        break;
 }
draw_border (13,24,15,54);
goto_xy(14,25);

 switch (rtn.second)
 {
 case 0:
        printf("No data error present");
        break;
 case 1:
        printf("Bus error");
        break;
 case 2:
        printf("Memory test error");
        break;
 case 17:
        printf("Arithmetic error");
        break;
 case 18:
        printf("Programming errror");
        break;
 case 19:
        printf("Invalid op code");
        break;
 case 20:
        printf("Invalid data");
        break;
 case 21:
        printf("Invalid port number");
        break;
 case 22:
        printf("Stack error");
        break;
 case 64:
        printf("Point out of workspace error");
        break;
   }

draw_border(19,24,21,54);
goto_xy(20,25);
switch (gotit.rejstat)
```

```
    {
     case 16:
           printf("Record format error");
           break;
     case 21:
           printf("Invalid port number");
           break;
     case 32:
           printf("Undefined Record");
           break;
     case 48:
           printf("Improper startup sequence");
           break;
     case 64:
           printf("Point out of workspace");
           break;
     case 80:
           printf("Insufficient memory");
           break;
     case 81:
           printf("Invalid robot type");
           break;
     case 83:
           printf("Invalid application number");
           break;
     case 96:
           printf("Invalid identifier sent before N
record");
           break;
     case 112:
           printf("Xoff time out (30 seconds)");
           break;
     case 128:
           printf("Manipulator power off");
           break;
     case 0:
           printf("No reject status");
           break;
     }
     while ( !kbhit());
    }

int downloadrob (number,part,name)     /* downloading of
program  */
int *number,*part;
char *name;
{                                          /*    function */
FILE *notes;
char cool;
```

```
unsigned char send_char, send_char1,send_char2;
unsigned comerror;
int i,f;

if ( *number != (1))
  {
   return 1;
  }

     f = *part;
     if ( f > 5)
      {
      return 2;
      }
     draw_border(17,24,19,54);
     switch (f)
            {
            case 1:
            goto_xy(18,25);
            printf("Downloading to partition 1");
            PARTT1 =1;
            send_char1 ='1';
            send_char2 ='B';
            break;
            case 2:
            goto_xy(18,25);
            printf("Downloading to partition 2");
            PARTT2 = 1;
            send_char1 ='2';
            send_char2 ='C';
            break;
            case 3:
            goto_xy(18,25);
            printf("Downloading to partition 3");
            PARTT3 =1;
            send_char1 ='3';
            send_char2 ='D';
            break;
            case 4:
            goto_xy(18,25);
            printf("Downloading to partition 4");
            PARTT4 = 1;
            send_char1 = '4';
            send_char2 = 'E';
            break;
            case 5:
            goto_xy(18,25);
            printf("Downloading to partition 5");
            PARTT5 =1;
            send_char1 = '5';
            send_char2 = 'F';
```

```
            break;
            case 40:
            break;
           default:
            break;
            }
     i =0;
     if ((notes = fopen(name,"r")) != NULL)
     {
      while ((send_char=getc(notes)) && (feof (notes) ==0))
        {
            i++;

            if (i == 5)
            {
            send_char = send_char1;
            }
            if (i ==23)
            {
            send_char = send_char2;
            }
            switch (send_char)
          {
          case 0x4E:
             sendcom(send_char);
             ack();
             break;
          case 0x0A:
             send_char = 0x0D;
             sendcom(send_char);
             send_char = 0x0A;
             sendcom(send_char);
             ack();
             break;
          default:
             sendcom(send_char);
             break;
          }

        }
     fclose (notes);
     return 0;
     }
     else
     return 3;
}

void return_home()                          /*Return home routine */
 {
 unsigned char send_char;
```

```
unsigned comerror;
unsigned char receive_char;

 send_char = 0x58;
 sendcom (send_char);
 receivecom (&receive_char,&comerror);
  if (receive_char == 0x13) {
      }
 receivecom (&receive_char, &comerror);
      if (receive_char == 0x11)
      {
      sendcom ( 0x31 );
      sendcom ( 0x31 );
      sendcom ( 0x0D );
      sendcom ( 0x0A );
      receivecom (&receive_char,&comerror);
      do
      {
        goto_xy(19,25);
        printf ("returning home");
        receivecom (&receive_char, &comerror);
        if ( receive_char == 0x04 )
            {
          goto_xy(19,25);
          printf ("error in return home ");
          return;
            }
      }
      while (receive_char != 0x11);
      goto_xy(19,25);
      printf("Home reached");
      return;
      }
}

int start_cycle(number,partition)       /*Start cycle
routine */
int *number;
int *partition;
 {
   int i,rtn;
   unsigned char send_char;
   unsigned comerror;
   unsigned char receive_char;
   if (*number != 1)
     {
      return 1;
     }
   i = *partition;
   switch (i)
```

```
    {
    case 1:
      rtn =application1 ();
      if (rtn ==1)
      return 4;
      break;
    case 2:
      rtn =application2 ();
      if (rtn ==1)
      return 4;
      break;
    case 3:
      rtn =application3 ();
      if (rtn ==1)
      return 4;
      break;
    case 4:
      rtn =application4 ();
      if (rtn ==1)
      return 4;
      break;
    case 5:
      rtn =application5 ();
      if (rtn ==1)
      return 4;
      break;
    default:
      break;
    }

  send_char = 0x58;
  sendcom (send_char);
  receivecom (&receive_char,&comerror);
  if (receive_char == 0x13) {
     }
  receivecom (&receive_char, &comerror);
     if (receive_char == 0x11) {
     sendcom ( 0x32 );
     sendcom ( 0x32 );
     sendcom ( 0x0D );
     sendcom ( 0x0A );
     receivecom (&receive_char,&comerror);
     if (receive_char == 0x06) {
     }
     STATEROBOT = 2;
     return 0;
     }
     else  {
     printf ("error in start cycle function");
     return 2;
     }
```

```
}

int stop_cycle(mcnumber) /*Stop cycle routine */
int *mcnumber;
 {
 unsigned char send_char;
 unsigned comerror;
 unsigned char receive_char;

  if ( *mcnumber != 1 )
     return 1;
  send_char = 0x58;
  sendcom (send_char);
  receivecom (&receive_char,&comerror);
  receivecom (&receive_char, &comerror);
      if (receive_char == 0x11) {
      sendcom ( 0x32 );
      sendcom ( 0x33 );
      sendcom ( 0x0D );
      sendcom ( 0x0A );
      receivecom (&receive_char,&comerror);
      if (receive_char == 0x13) {
      goto_xy(19,24);
      printf (" Stopping Cycle ");
      receivecom (&receive_char, &comerror);
      if (receive_char == 0x11) {
      goto_xy(19,24);
      printf("Stopped cycle");
      STATEROBOT = 0;
      return;
      }
     }
     }
      else {
      printf ("error in stop cycle function\n");
      return;
      }
 }

void flushpart()
{
int i = 0;
int f;
unsigned char send_char,send_char1,send_char2;
unsigned comerror;
FILE *notes;
extern char ptr[9];
   for (f=1;f<6;f++)
      {
     draw_border(21,24,23,54);
     switch (f)
```

```
            {
            case 1:
            goto_xy(22,25);
            printf("Flushing partition 1");
            send_char1 ='1';
            send_char2 ='B';
            break;
            case 2:
            goto_xy(22,25);
            printf("Flushing partition 2");
            send_char1 ='2';
            send_char2 ='C';
            break;
            case 3:
            goto_xy(22,25);
            printf("Flushing partition 3");
            send_char1 ='3';
            send_char2 ='D';
            break;
            case 4:
            goto_xy(22,25);
            printf("Flushing partition 4");
            send_char1 = '4';
            send_char2 = 'E';
            break;
            case 5:
            goto_xy(22,25);
            printf("Flushing partition 5");
            send_char1 = '5';
            send_char2 = 'F';
            break;
           default:
            break;
            }

if ((notes = fopen(ptr,"r")) != NULL)
   {
    while ((send_char=getc(notes)) && (feof (notes) ==0))
     {
            i++;

            if (i == 5)
            {
            send_char = send_char1;
            }
            if (i ==23)
            {
            send_char = send_char2;
            }
            switch (send_char)
          {
```

```
            case 0x4E:
               sendcom(send_char);
               ack();
               break;
            case 0x0A:
               send_char = 0x0D;
               sendcom(send_char);
               send_char = 0x0A;
               sendcom(send_char);
               ack();
               break;
            case 0x45:
               sendcom(send_char);
               send_char=0x4E;
               sendcom (send_char);
               break;
             default:
               sendcom(send_char);
               break;
            }

          }
      fclose (notes);
        }
      }
}



void autostart()                         /*Auto execution mode */
 {
 unsigned char send_char;
 unsigned comerror;
 unsigned char receive_char;

  send_char = 0x58;
  sendcom (send_char);
  receivecom (&receive_char,&comerror);
  receivecom (&receive_char, &comerror);
      if (receive_char == 0x11) {
      sendcom ( 0x32 );
      sendcom ( 0x30 );
      sendcom ( 0x0D );
      sendcom ( 0x0A );
      receivecom (&receive_char,&comerror);
      if (receive_char == 0x06) {
      goto_xy(19,25);
      printf("Set on Auto mode");
      STATEROBOT = 1;
      while (!kbhit());
```

```
       return;
       }
       }
       else  {
       goto_xy(19,25);
       printf ("error in auto mode ");
       return;
       }
 }


int application1()                  /*Select Application 1
*/
 {
 unsigned char send_char;
 unsigned comerror;
 unsigned char receive_char;

  if ( PARTT1 != 1 )
     return 1;
  send_char = 0x58;
  sendcom (send_char);
  receivecom (&receive_char,&comerror);
  printf ("%c\n",receive_char);
   if (receive_char == 0x13) {

       printf ("X off\n");
       }
  receivecom (&receive_char, &comerror);
       if (receive_char == 0x11) {
       sendcom ( 0x33 );
       sendcom ( 0x31 );
       sendcom ( 0x0D );
       sendcom ( 0x0A );
       receivecom (&receive_char,&comerror);
       if (receive_char == 0x06) {
       goto_xy(22,42);
       printf (":Partition 1");
       return 0;
       }
       }
       else  {
       printf ("error in  function");
       return;
       }
 }


int application2()                  /*Select Application 2
*/
```

```
{
unsigned char send_char;
unsigned comerror;
unsigned char receive_char;

 if ( PARTT2 != 1)
     return 1;
 send_char = 0x58;
 sendcom (send_char);
 receivecom (&receive_char,&comerror);
 printf ("%c\n",receive_char);
  if (receive_char == 0x13) {

     }
 receivecom (&receive_char, &comerror);
     if (receive_char == 0x11) {
     sendcom ( 0x33 );
     sendcom ( 0x32 );
     sendcom ( 0x0D );
     sendcom ( 0x0A );
     receivecom (&receive_char,&comerror);
     if (receive_char == 0x06) {
     goto_xy(22,42);
     printf (":Partition 2");
     return 0;
     }
     }
     else  {
     printf ("error in  function\n");
     return;
     }
}


int application3()                    /*Select Application 3
*/
{
unsigned char send_char;
unsigned comerror;
unsigned char receive_char;

 if ( PARTT3 != 1)
   return 1;
 send_char = 0x58;
 sendcom (send_char);
 receivecom (&receive_char,&comerror);
 receivecom (&receive_char, &comerror);
     if (receive_char == 0x11) {
     sendcom ( 0x33 );
     sendcom ( 0x33 );
     sendcom ( 0x0D );
```

```
      sendcom ( 0x0A );
      receivecom (&receive_char,&comerror);
      if (receive_char == 0x06) {
      goto_xy(22,42);
      printf ("Partition 3\n");
      return;
      }
      }
      else  {
      printf ("error in  function\n");
      return;
      }
 }


int application4()                     /*Select Application 4
*/
 {
 unsigned char send_char;
 unsigned comerror;
 unsigned char receive_char;

  if ( PARTT4 != 1)
  return 1;
  send_char = 0x58;
  sendcom (send_char);
  receivecom (&receive_char,&comerror);
  printf ("%c\n",receive_char);
   if (receive_char == 0x13) {

      printf ("X off\n");
      }
  receivecom (&receive_char, &comerror);
      if (receive_char == 0x11) {
      sendcom ( 0x33 );
      sendcom ( 0x34 );
      sendcom ( 0x0D );
      sendcom ( 0x0A );
      receivecom (&receive_char,&comerror);
      if (receive_char == 0x06) {
      goto_xy(22,42);
      printf (":Partition 4\n");
      return 0;
      }
      }
      else  {
      printf ("error in  function\n");
      return;
      }
 }
```

```
int application5()            /*Select Application 5 */

 {
 unsigned char send_char;
 unsigned comerror;
 unsigned char receive_char;

 if ( PARTT5 != 1)
    return 1;
  send_char = 0x58;
  sendcom (send_char);
  receivecom (&receive_char,&comerror);
  printf ("%c\n",receive_char);
   if (receive_char == 0x13) {

      printf ("X off\n");
      }
  receivecom (&receive_char, &comerror);
      if (receive_char == 0x11) {
      sendcom ( 0x33 );
      sendcom ( 0x35 );
      sendcom ( 0x0D );
      sendcom ( 0x0A );
      receivecom (&receive_char,&comerror);
      if (receive_char == 0x06) {
      goto_xy(22,42);
      printf (":Partition 5\n");
      return;
      }
      }
      else  {
      printf ("error in  function\n");
      return;
      }
 }

MC_STAT mc_status(mcnumber)
int *mcnumber;
{
int i,bfirst,bsecond,p;
int int3,int4,int5,int6;
MC_STAT thevar;
char s[21],ch;
unsigned send_char;
unsigned comerror;
unsigned char receive_char;

 /*   if ( *mcnumber != 1)
```

```
      return 3; */
    send_char = 0x52;
    sendcom (send_char);
    while (receive_char != 0x11)
    {
    receivecom (&receive_char,&comerror);
    }
      sendcom(0x30);
      sendcom(0x31);
      sendcom(0x0D);
      sendcom(0x0A);
    while (receive_char != 0x0A)
    {
    receivecom(&receive_char,&comerror);
    }
    i = 0;
    while (ch != 0x0A)
    {
    ch = _bios_serialcom (_COM_RECEIVE,0,0);
    s[i] =ch;
    i++;
    }
    sendcom (0x06);
    receivecom(&receive_char, &comerror);
    while (receive_char != 0x0D )
    {
     receivecom(&receive_char, &comerror);
    }
    receivecom (&receive_char, &comerror);
    sendcom(0x06);
    int3 =(s[6]-48);
    int4 =(s[7]-48);
    int5 =(s[8]-48);
    int6 =(s[9]-48);
    bfirst = (int4+int3*16);
    bsecond = (int6+int5*16);

    thevar.first=bfirst;
    thevar.second=bsecond;
    thevar.staterobot=STATEROBOT;

    return (thevar);

}


REJ_STAT reject_status()
{
int i,bfirst,bsecond,p;
int int3,int4;
char s[21],ch;
```

```
REJ_STAT mark;
unsigned send_char;
unsigned comerror;
unsigned char receive_char;

    send_char = 0x52;
    sendcom (send_char);
    while (receive_char != 0x11)
    {
    receivecom (&receive_char,&comerror);
    }
       sendcom(0x30);
       sendcom(0x32);
       sendcom(0x0D);
       sendcom(0x0A);
    while (receive_char != 0x0A)
    {
    receivecom(&receive_char,&comerror);
    }
    i = 0;
    while (ch != 0x0A)
    {
    ch = _bios_serialcom (_COM_RECEIVE,0,0);
    s[i] =ch;
    i++;
    }
    sendcom (0x06);
    receivecom(&receive_char, &comerror);
    while (receive_char != 0x0D )
    {
     receivecom(&receive_char, &comerror);
    }
    receivecom (&receive_char, &comerror);
    sendcom(0x06);
    int3 =(s[6]-48);
    int4 =(s[7]-48);
    bfirst = (int4+int3*16);
    mark.rejstat=bfirst;
    return(mark);
 }

void di_do_status()
{
int s[6],X,i;
int input_1,input_2,output_1,output_2;
unsigned send_char;
unsigned comerror;
unsigned char receive_char;

    send_char = 0x52;
    sendcom (send_char);
```

```
    receivecom (&receive_char,&comerror);
    if (receive_char == 0x13) {
         printf ("X off\n ");
         }
    receivecom(&receive_char, &comerror);
    while (receive_char != 0x11) {
       printf ("Waiting for Xon\n");
       }
    if (receive_char == 0x11) {
       sendcom(0x31);
       sendcom(0x30);
       sendcom(0x0D);
       sendcom(0x0A);
       }
    receivecom(&receive_char, &comerror);
       while (receive_char != 0x44);
    for (i = 0;i <= 5; i++){
       receivecom(&receive_char, &comerror);
       s[i] = receive_char;
       }
    receivecom(&receive_char, &comerror);
    while (receive_char != 0x0D);
    receivecom(&receive_char, &comerror);
    while (receive_char !=0x0A);
    X = ((s[1] + s[2] + s[3] + s[4]) % 256);
    if (s[5] != X) {
     printf ("Check Sum error\n");
     exit(1);
     }
    sendcom (0x06);
    receivecom(&receive_char, &comerror);
    if (receive_char == 0x45 ) {
     receivecom(&receive_char, &comerror);
     if (receive_char == 0x47 ){
        printf("Good Data,End of Data\n");
        }
    }
    receivecom(&receive_char, &comerror);
    while (receive_char != 0x0D);
    receivecom(&receive_char, &comerror);
    while (receive_char !=0x0A);
    sendcom(0x06);
    input_1 = s[1];
    input_2 = s[2] ;
    output_1 = s[3] ;
    output_2 = s[4] ;
    return;
}

void ack()
 {
```

```
 unsigned char receive_char;
 unsigned comerror;
   receivecom (&receive_char,&comerror);
   if (receive_char == 0x06)
       {
       return;

       }
   else
       {
       printf ("%c",receive_char );
       exit(1);
       }
 }


unsigned getcomstatus ()              /* COM status routine
*/
  {
    return(_bios_serialcom (_COM_STATUS, WHICHCOMPORT, 0));
  }



unsigned initcom (comdata )     /* COM initialization
routine    */
  {
    if (WHICHCOMPORT > 1)             /* Check for valid COM
port if only 1 COM     */
                                /* port is installed, put 0
instead of 1      */
      {
      printf ("Invalid COM port %u\n", WHICHCOMPORT + 1);
      return(0xFFFF);      /* Set error flag if port number
out of range    */
      }
    else
      return(_bios_serialcom (_COM_INIT, WHICHCOMPORT,
comdata));
  }

unsigned sendcom (com_char)       /* COM send routine
*/
  unsigned char com_char;
  {
  unsigned comerror;
    do {
     comerror = _bios_serialcom (_COM_SEND, WHICHCOMPORT,
com_char) & 0xFF00;
```

```
     if (!(comerror & 0x8000))
         return(comerror&0x00FF);
     printf("DTR down\nPlease get ROBOT online \npress any
key to continue. . .\n");

     } while (!kbhit());
  }



receivecom ( com_char, comerror)    /* COM receive routine
*/
 unsigned *comerror;
  unsigned char *com_char;
  {
    while ((com_status & 0x2000) < 0x2000)          /*
Check Transmission    */
      com_status = getcomstatus ();        /* Holding
Register      */
    *com_char  = 0;
    *comerror = 0;
                    /* Repeat until a character was received
or   */
                    /* an error occurs.  If time-out should
be    */
                    /* enabled change 0x1E00 to 0x9E00
*/
    while (((*comerror & 0x1E00) < 0x0200) && (*com_char <
0x01))
      {
      *comerror = _bios_serialcom (_COM_RECEIVE, 0, 0);
                    /* Character is returned in low byte
*/

      *com_char  = *comerror & 0x00FF;
                      /* Error is returned in high    byte
*/
      *comerror = *comerror & 0xFF00;
      }
  }
```

APPENDIX B: Dyna operations

The DYNA machine has an RS 232C interface which allows the user to interface the controller with an external computer or peripheral communication link. The user may:

UPLOAD a program from the controller to the computer (to be stored or printed out).

PROGRAM LOAD a program from the computer to the controller. The program cannot exceed 900 lines of code.

LINE EXECUTE a program from the computer to the controller a line at a time. This allows infinite length but the execution is slow due to the communication overhead.

PROGRAM EXECUTE allows the computer to download blocks of program at the time of execution.

DYNA OPERATION PROCEDURE

Step 1: Turn on the power to the Dyna Milling machine and let the machine do the backlash check.

Step 2: Clear the controller memory before transferring the program from the cell controller.

Step 3: Setup the Dyna to get ready for communications.

(1) Press "LINE No" key

(2) Press "SHIFT" key

(3) Press "Read/Write" key

At this moment the LCD on the Dyna controller display "UPLOAD".

Step 4: As DOWNLOAD is to be selected, press "No" key on the Dyna controller and "DOWNLOAD" will be displayed on the controller LCD.

Step 5: Press "Yes" and the display will show "LINE EXECUTE". Press "No" and the display will show "PROGRAM EXECUTE".

Step 6: Press the "Yes" key and this will bring the user into the program block execution mode. The LCD will display "READY" until the first block of code is downloaded into the controller and then display "setup >zcxyu" is shown.

STEP 7: Now the user should proceed to setup the reference zero by jogging the axis. Once the reference zero has been set, press the "Next" key. Now the controller waits for the next block of code.

Whenever "PROGRAM EXECUTE" mode is applied to execute programs which are more than 900 lines, the program must be divided into several blocks. Each block cannot be more than 900 lines and the last instruction for each block must be

"SKIP TO 900" and the instruction "900 (blank)" must be after the "SKIP TO 900" instruction. The Dyna controller will download the program and begin execution until both "SKIP TO 900" and "900 (BLANK)" are reached. After this block is executed the Dyna controller will be "READY" to download and execute next block of program when sent from the computer. This will continue until "END" statement is received .

APPENDIX C: Examples of Dyna Programs

Example of a Setup Program

```
000 START INS 01
001 TD = .125
002 FR XYZ = 10.0
003 SETUP >zcxyu
004 SKIP TO 900
900
```

Examples of Milling Programs

```
000 GO X 1.000
001    Y 1.000
003 GO X 1.100
004 GR X 2.000
005 SKIP TO 900
900
```

```
000 GO X .5
001 GO Y .7
002 GR X .2
003 GO X 1.000
004 GO Y 2.000
005 SKIP TO 900
900
```

Example of a End Program

```
000  END
```

APPENDIX D: Example of a compiled AML/E program

```
N091175452020202020208B
D10001670110000 3D711100013D7211000219
D103D731100033E62230101001F04110007C4
D102300373D0203130004120005 0C00040CE6
D1000050D000701FFE701001F230010000558
D1023041000070 2FFD7602301045223003 84B
D103D046223013E033E0100263E2322100000
D100402FFDA232610000402FFD2232A10006C
D100402FFCA232E10000402FFC2070369056F
D0B3E02FFDA02FFD702FFD409CF
L044001000041
W100000000000000000000000000000000000
W100000000000000000000000000000000000
W100000000000000000000000000000000000
W100000000000000000000000000000000000
W100000000000000000000000000000000000
W100000000000000000000000000000000000
W100000000000000000000000000000000000
W100000000000000000000000000000000000
W10000000000000000087FA00000A95C000E0
W10000000000000000087FA000009C8000052
W100000000000000000008FA000009C80000D3
W100000000000000000008FA00000A95C00061
W08000000000000000000
EG
```

APPENDIX E: _BIOS_SERIALCOM

Summary

The _BIOS_SERIALCOM() routine can be used by including the bios header file. The section below describes the parameter types taken by _BIOS_SERIALCOM() routine.

```
# include < bios.h >
unsigned _bios_serialcom(service, serial_port, data)
unsigned service            /* Communication port */
unsigned serial_port        /* Serial port to use */
unsigned data               /* Port configuration bits */
```

Description

The _bios_serialcom routine uses INT 0x14 to provide serial communication services. The serial_port argument is set to zero for COM1, to 1 for COM2, and so on. The service arguments can be set to one of the following manifest constants.

_COM_INIT   Sets the port to the parameters specified in the data argument,

_COM_SEND   Transmits the data characters over the

	selected serial port,
_COM_RECEIVE	Accepts the input character from the selected serial port or
_COM_STATUS	Returns the current status of the selected serial port.

The data argument is ignored if the service is set to _COM_RECEIVE or _COM_STATUS. The data argument for _COM_INIT is created by ORing together one or more of the following constants:

Constant	Meaning
_COM_CHAR7	7 data bits
_COM_CHR8	8 data bits
_COM_STOP1	1 stop bit
_COM_STOP2	2 stop bit
_COM_NOPARITY	No parity
_COM_EVENPARITY	Even parity
_COM_ODDPARITY	Odd parity
_COM_110	110 baud
_COM-150	150 baud
_COM_300	300 baud
_COM_600	600 baud
_COM_1200	1200 baud
_COM_2400	2400 baud
_COM_4800	4800 baud
_COM_9600	9600 baud

The default value of data is 1 stop bit, no parity bit, and

110 baud.

Return value

The function returns a 16 bit integer whose high-order byte contains status bits. The meaning of the low-order byte varies, depending on the service values. The high-order bits are as follows:

Bit	Meaning if set
15	Timed out
14	Transmission shift register empty
13	Transmission hold register empty
12	Break detected
11	Framing error
10	Parity error
9	Overrun error
8	Data error

When service is _COM_SEND, bit 15 will be set if data could not be sent. When service is _COM_RECEIVE, the byte read will be returned in the low order bits if the call is successful. If an error occurs, at least one of the high order bits will be set. When service is _COM_INIT or _COM_STATUS, the low order bits are defined as follows:

Bit	Meaning if set
7	Receive-line signal detected

6	Ring indicator
5	Data-set ready
4	Clear to send
3	Change in receive line signal detected
2	Trailing edge ring indicator
1	Change in data set ready status
0	Change in clear to send status

APPENDIX F: Communication protocol of IBM 7545/7547 robots

The robot controller communicates with the host computer using a transaction based protocol, with the host controlling all transactions. The transactions consist of:

- *identifier
- *records
- *Optional collection of identifiers, records, requests and responses

The different identifiers specify different types of functions that can be performed by the communications package. They can be classified into five categories:

R - This is the Read record. It instructs the controller to send back certain information requested by the host computer. It will be described in detail later.

X - This identifier is associated with the remote functions available. It instructs the controller to execute a remote function.

N - This identifier informs the controller that a compiled program has to be downloaded.

D - All data to be communicated to and from the controller is sent in a pre-specified format, the D record. What follows a D identifier is usually a set of data values. As the D is always sent as an answer to a request, and never in isolation, it can be considered as a part of the entire record.

E - This identifier is sent to identify the end of a good data transmission.

A standard is made up of the following components:

id bc data cs CrLf

"id" represents the appropriate identifier.
"bc" represents the number of bytes of data to follow.
"data" is the data values in hex format.
"cs" is the check sum (modulo 256) of the data.
"CrLf" represents the carriage return and line feed to signify the end of transmission of the record.

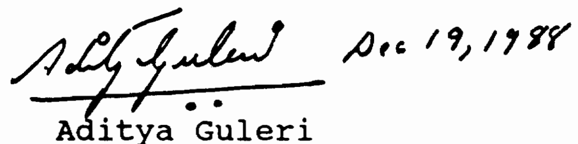
There is a set of restrictions to be kept in mind when data is to be transmitted. All numeric data must be represented in hexadecimal code. All data is transmitted as hexadecimal representation of ASCII code. As the system deals in ASCII code, standard ASCII control characters can be used to control transactions.

The hardware interface requires the use of a RS-232

interface for communicating between the host computer and the robot controller. The protocol requires the use of a 4800 baud rate, even parity, full duplex transmission, 7 data bits and 2 stop bits.

VITA

Aditya Guleri was born on February 11th, 1965 in Chandigarh, India. After completing Pre-Engineering, he enrolled at Punjab Engineering College, Chandigarh in August 1982. He graduated with honors in July 1986 with a Bachelor of Engineering in Electrical Engineering. After working in DCM-Tandy Corp. for a period of six months as a Management Trainee, he enrolled at Virginia Tech where he is currently pursuing a Master of Science Degree in Industrial Engineering and Operations Research. After this, he plans to work as a Computer Aided Manufacturing Engineer for LSI Logic Corp. in Milpitas, California.



Aditya Guleri