Structure of a Flexible Manufacturing Protocol for Design and Control of a Vertical Machining Cell

by: S. Chen
Advisor: D.K. Anand

M.S. 88-5
Formerly TR 88-59

ABSTRACT

Title of Thesis: Structure of a Flexible Manufacturing Protocol for Design and Control of a Vertical Machining Cell

Sujen Chen, Master of Science, 1988

Thesis directed by: Dr. D. K. Anand
Professor
Mechanical Engineering

The thesis presented here develops the Flexible Manufacturing Protocol (FMP) for use in CIM (Computer-integrated Manufacturing). The philosophy of the FMP is discussed, as well as the FMP structure. A vertical machining cell was built for automated machining of prismatic parts based on FMP concepts. This cell uses two computers to simulate the responses of cell level components. The system hierarchy and software structure are described in detail as well as the operation schemes. The interfacing / integration problems encountered during the development of the cell are also discussed. An example of how the FMP works with the vertical machining cell is included. The successful demonstration of the FMP-based system capability further reflects the need for FMP and how it is going to benefit the CIM industry.

STRUCTURE OF A FLEXIBLE MANUFACTURING PROTOCOL FOR DESIGN AND CONTROL OF A VERTICAL MACHINING CELL

by

Sujen Chen

Thesis submitted to the Faculty of the Graduate School
of the University of Maryland in partial fulfillment
of the requirements for the degree of
Master of Science
1988

ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to my thesis advisor Dr. D. K. Anand who presented me with this interesting and challenging field and helped me to see it through. Without Dr. Anand's support, financially and academically, it would be more difficult to accomplish this work. The support from the Systems Research Center of National Science Foundation is also appreciated.

I would like to thank Dr. J. Kirk, whose guidance and support through out my research were invaluable. In addition, I wish to thank Dr. M. Anjanappa and M. Unger for their suggestions and comments. I also like to thank Dr. C. Sayre for serving on my thesis committee.

Finally, I would like to thank my parents, for their sacrifice and patience during the past years, and my wife for her support through my Master's program.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 The History of Automation

The term "automation" has many definitions. Apparently, it was used first in the 1950's to mean automatic materials handling, particularly equipment used to unload and load stamping equipment [41]. It has now become a general term referring to the technologies concerned with the application of complex mechanical, electronic, and computer based systems in the operation and control of production [37].

Numerical Control (NC) can be considered as a form of programming automation. Shortly after the first demonstration of the NC prototype in 1952, the potential usefulness of the NC concept was proven. In the early 1960's, NC machines were installed at various factories to achieve machining automation. These NC machines accepted programmed tool paths in the form of punched tapes prepared on computers, and hence a different part needed only a new punched tape.

Automation was also introduced for mass production due to its positive cost performance effect on investment.

After most shops were automated by mid 1960's, it was realized that the mass production accounted for only 25% of the total production in the United States. The other 75% of production is of the batch type which imposes difficulties for automation because of lack of flexibility. The need to automate batch production, however, was essential to improving the productivity [14].

With the successful development of Direct Numerical Control (DNC) and Computer Numerical Control (CNC) machines (later versions of NC), the gap between the inflexible, mass production type machines and highly flexible, general purpose machine tools was bridged. It also created entirely new concepts in manufacturing, such as certain routing operations that previously were very difficult to accomplish. Coupled with the introduction of the Automatic Tool Changing (ATC) mechanism, the Automated Guided Vehicle (AGV), and robots, the automation of batch type manufacturing first become feasible. In early 1980's, several such systems were installed to machine parts such as engine blocks, jet engine parts, etc [25].

1.2 CAD/CAM and CIM

CAD/CAM is a term which means Computer-Aided Design and Computer-Aided Manufacturing. "It is the technology concerned with the use of digital computers to perform certain functions in design and production.", according to Mikell P. Groover, "CAD can be defined as the use of computer systems to assist in the creation, modification, analysis, or optimization of a design. CAM can be defined as the use of computer systems to plan, manage, and control the operations of a manufacturing plant.".

When Computer-Aided Design (CAD) was getting under way in the mid-1960's, large mainframes were needed to supply the computing power that the graphic techniques required. As minicomputers and microcomputers were introduced in 1970's, more computing power, lower costs, and smaller sizes were provided and stand-alone systems became feasible. At present, mainframe-based systems, stand-alone systems and various combinations are in use, and CAD/CAM operations are performed on a batch basis [1]. Because of the important role of computers, these developments are often called Computer Integrated Manufacturing (CIM).

Today, however, CIM includes more than just CAD/CAM [29]. CIM has becoming much broader in scope, encompassing virtually everything that goes on in a manufacturing plant, including NC programming, manufacturing resource planning (MRP), shop floor communications and robotics,etc. CIM is also becoming widely recognized as a vital strategy to staying competitive in the increasing worldwide competition which requires high quality, economical manufacturing, and short delivery times [9].

1.3 The Need for FMC and FMS

With CIM as the strategy, the brief cycles of innovation together with smaller batch sizes have forced the industries to introduce flexible, computer-aided automation in manufacturing [11]. The minimum unit capable of automatic operation having NC machine tools as its core, is called the Flexible Manufacturing Cell [7]. A Flexible Manufacturing System (FMS) is a self-contained interlinkage of cells [12]. Fig. 1.1 and fig. 1.2 show the typical configuration of a FMC and a FMS, respectively.

An investigation shows that the FMS is ideal for mid-volume manufacturing of a small number of different parts, while the FMC is ideal for small volume manufacturing of a large variety of parts (fig. 1.3) [42].

The first FMC with computer control and a fully automatic supply of tools was presented in 1983 at the EMO (European Machine Tools Exihibition) in Paris. More than one hundred FMS consisting of several NC machine tools, AGVs for work piece and/or tool transportation, and storage for workpieces, jigs, pallets and tools have been installed in Japan to automate batch-type machining operations [6].

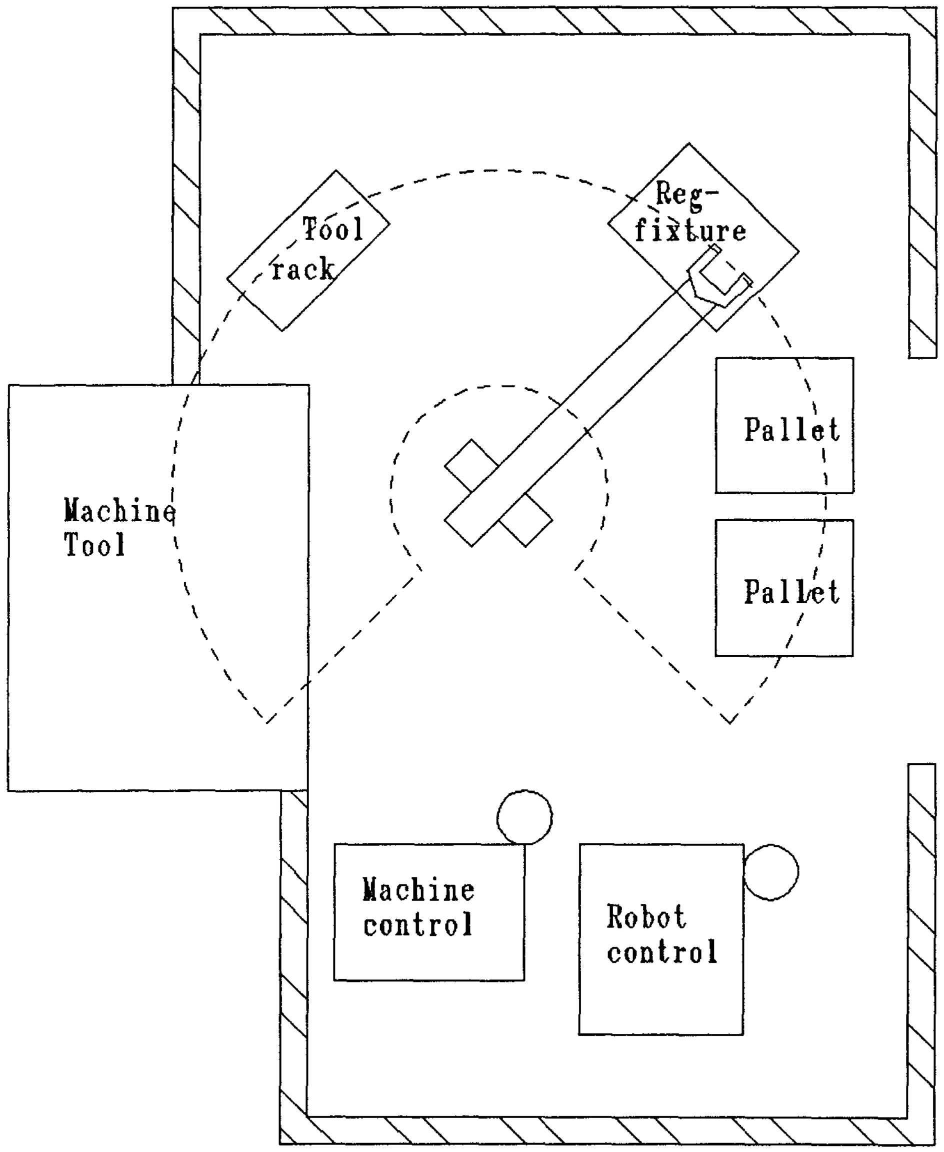


Figure 1.1 Flexible manufacturing cell

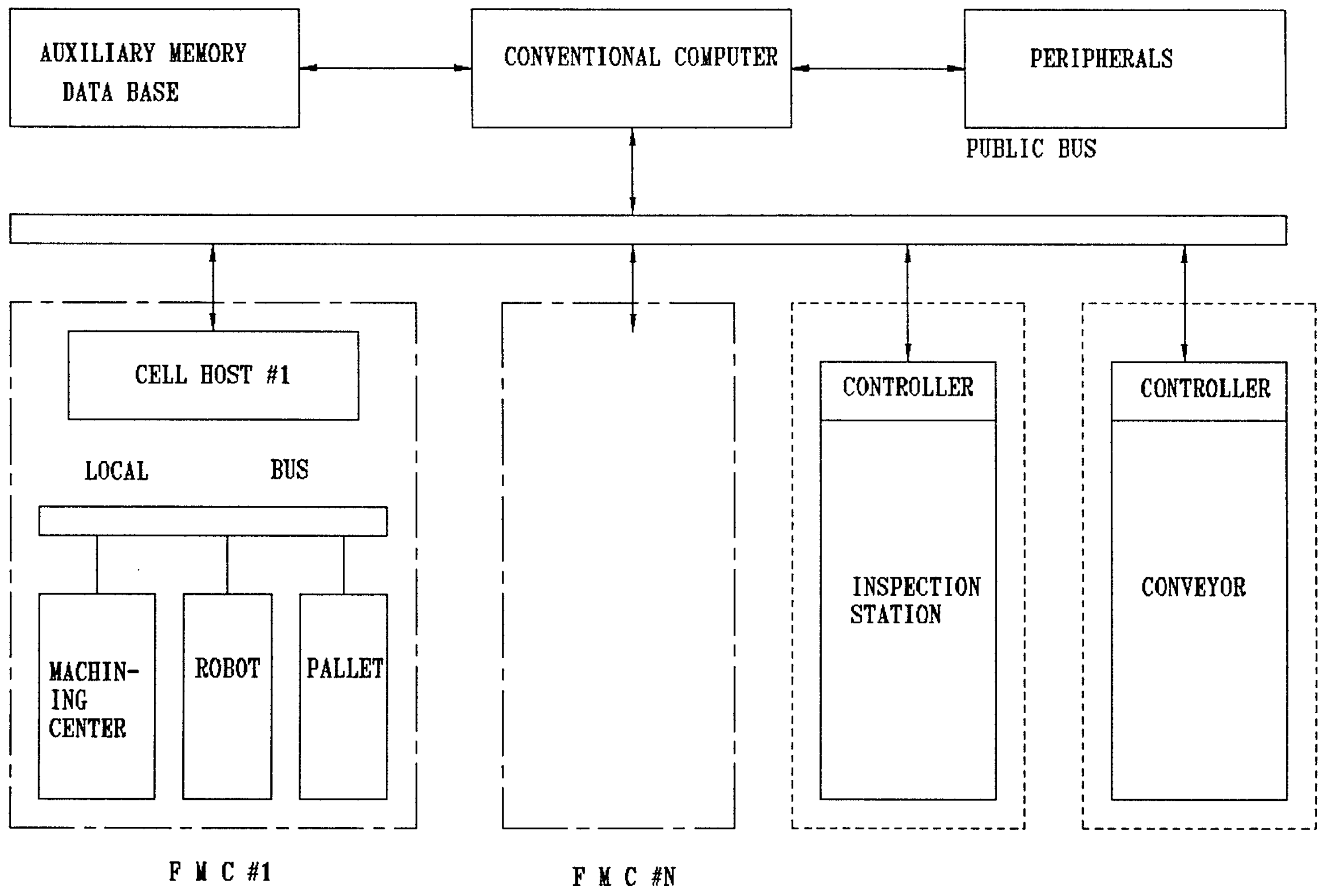


Figure 1.2 Flexible manufacturing system

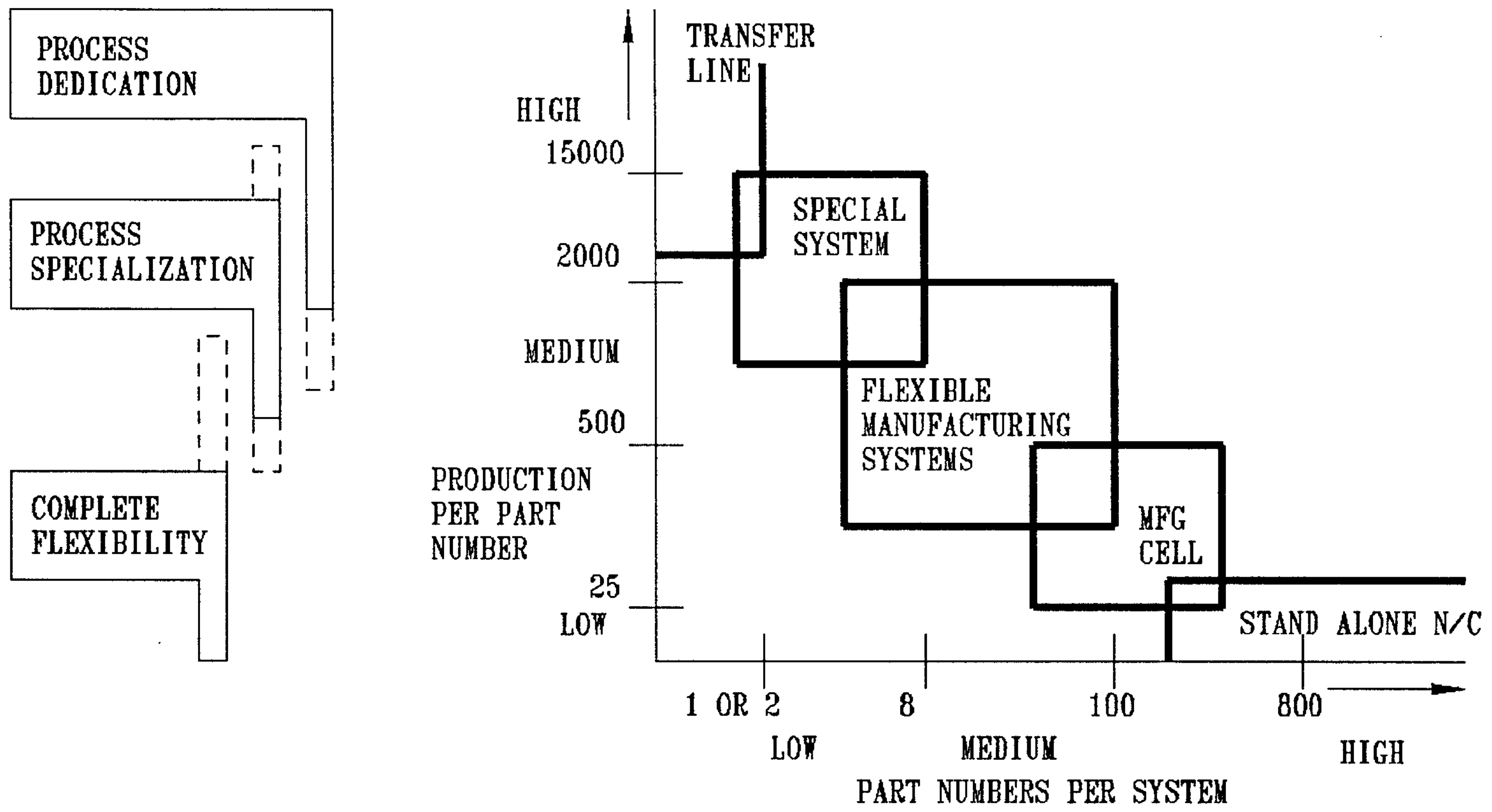


Figure 1.3 Manufacturing systems (courtesy Kearney and Trekker)

Measurable benefits include increased productivity and decreased work-in-process (WIP), total lead time, and product costs [5,6,8].

CHAPTER 2

THE FLEXIBLE MANUFACTURING PROTOCOL OF UMCP

2.1 Why Protocol?

The industry today is moving toward computer-aided production for staying competitive. At present, however, each computer-aided design or computer-aided manufacturing system typically deals with one isolated function. The system is planned and implemented by the management of that particular function. As the separate systems become more numerous, users are becoming aware of growing inefficiencies [1]. It is troublesome to learn the peculiarities of different systems; it is costly and inefficient to store often overlapping data for each separate system; it is a source of error to extract data from one system and enter it manually into the next one. Few examples of other types of inefficiencies are: a design database is created but can not be understood by another system that does the process planning; a user designed part is not manufacturable due to lack of understanding of the plant capabilities or plant facilities; inefficient design was made due to lack of manufacturing intuition; etc. These mentioned inefficiencies point to the fact that the computer-aided production methods existing today, such as CAD, CAM, CAP, CAQA, FMC, and FMS, should be unified into a single process

[15]. With this process, the data can be exchanged with other systems, the database contains varieties of information required throughout the manufacturing process, and intuitive processing to improve manufacturability can be brought into the design stage. The solution of this process is a protocol built with a well defined software structure and database management strategies, backboned with the important strategy: standardization, which ensures the flexibility of the protocol.

2.2 The Flexible Manufacturing Protocol

The study of the protocol was first initiated in 1986 at the Flexible Manufacturing Laboratory of the University of Maryland, College Park. A Flexible Manufacturing Protocol (FMP) was developed as the result of this research [4,14,19,24,42,47]. Shown in fig. 2.1 is a schematic diagram of the FMP for automated production within a cell. This includes all processes from design to part production including AGV and robot control.

2.2.1 Part Design

At the top of the FMP, the user is a designer who interacts with a CAD system in order to obtain a suitable design. The designer can design parts by two different approaches.

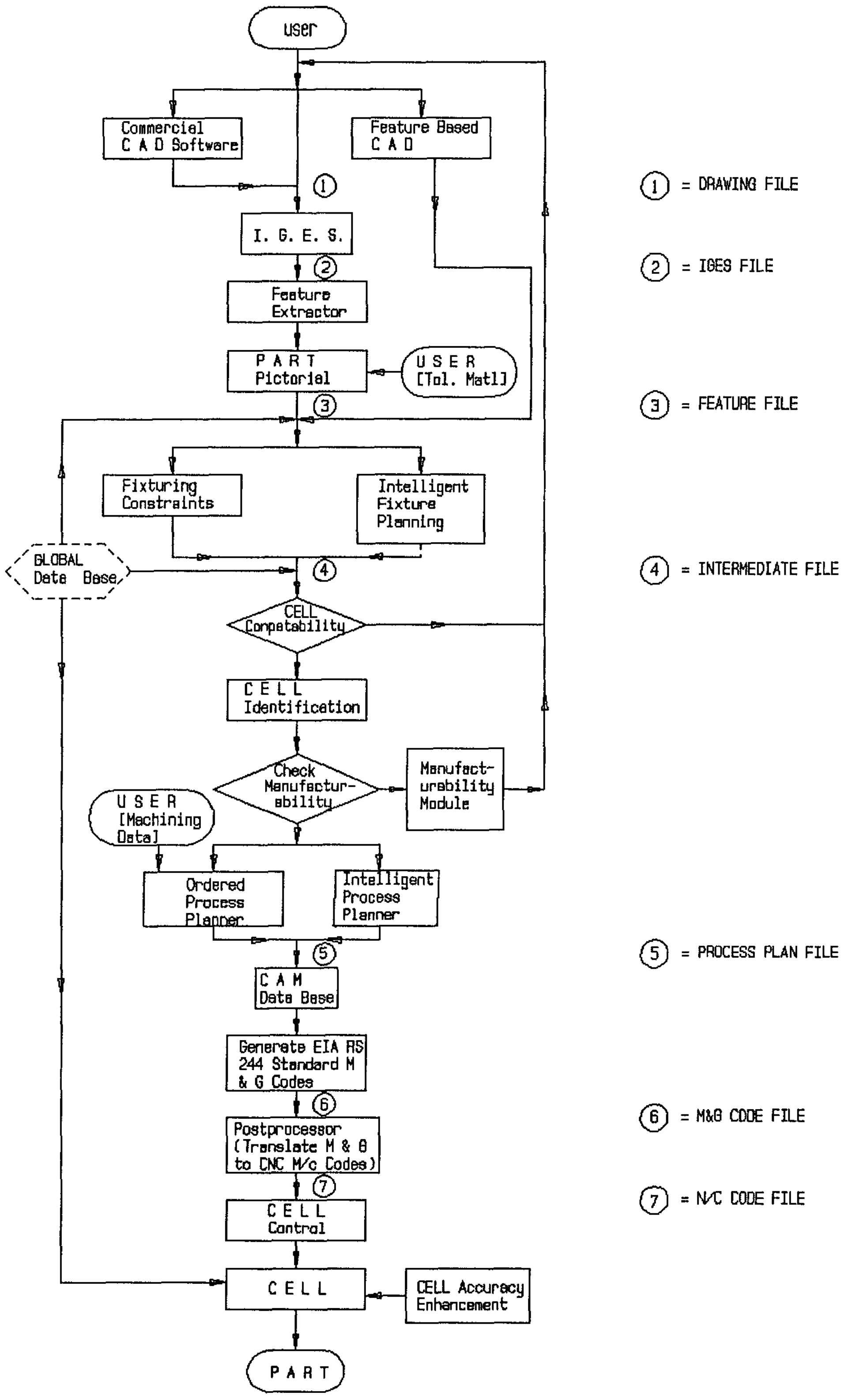


Figure 2.1 Flexible Manufacturing Protocol

The first is by using a commercial CAD system to create the design and stored the design in Initial Graphics Exchange Specification (IGES) format. The FMP then takes the IGES file for that particular part and decomposes it by a feature extractor into a collection of features such as grooves, pockets, slots, holes, etc. The user can also interact with the FMP to add the manufacturing information, such as tolerances, materials, and surface finish that can not be described by IGES. This information is stored in a feature file which consists of the geometry, topology, features, and manufacturing informations of a part.

The second approach is designing by using a special feature-based CAD system. The design data is stored in a feature file as the output of the feature-based design CAD system for subsequent processing.

2.2.2 Manufacturability

The next step is to evaluate the part and tolerance data in order to determine if production is compatible with the cells available to the user. Once the part has been evaluated as being compatible with an existing cell, the part is then checked for manufacturability. This includes a check for tolerances and geometrical machinability. The manufacturability tests will be conducted by reviewing individually each feature of the output of the feature extractor. This includes testing for manufacturability of sharp corners, tolerance, surface finish, interference between tool holders and workpiece, and materials encountered while drilling a hole, etc. If any of the above tests fail to qualify, the manufacturability module will then generate a message outlining the reasons for failure. The user will have to change the design of the part iteratively until the design passes the manufacturability tests.

2.2.3 Process Planning

Process planning can be defined as the process of determining the methods and the sequence of machining a workpiece to produce a finished part or component to satisfy the design specifications. Process planning typically consists of the following activities:

- Selection of processes and tools.
- Sequencing the processes.
- Identification of all non-machining elements and estimating the non-machining times.
- Selection of workpiece holding devices.
- Determination of proper cutting conditions and cutting times to machine the workpiece to specified dimensions.
- Determination and sequencing required shop floor activities for production.

In the FMP, there are two parallel paths which can be followed in producing the process plan. On the left side is the ordered process plan which uses a well established approach and is not discussed here. One of the unique features of FMP is the intelligent process planner based upon an "expert system approach". It takes the feature file as the input and outlines the required machining processes. The knowledge base for the expert system will include manufacturability information regarding each machinable feature and generate a set of processes for machining these features on an automated machining center.

2.2.4 NC Code Generation

The output of the process planner is processed through a pre-processor to generate EIA RS 244 standard M & G codes which are the US standards for cutting codes for NC/CNC machines. A post-processor then translates the standard M & G codes into the machine specific codes.

2.2.5 Cell Control

Based on the process plan generated by the process planner, the cell host will coordinate the activities in the shop floor to machine the part. Tasks involved during the manufacturing processes include:

- Transport the raw material into the cell via AGV.
- Transfer the material from AGV to the intelligent fixture on the machining center.
- Unload finished part from the machining center to the AGV.
- Transfer the finished part out of the cell via the AGV.

2.3 Standardization and Flexibility

It is clear that for the Flexible Manufacturing Protocol to operate in an efficient and highly flexible manner, a high degree of standardization must exist throughout the system. Notice in fig. 2.1 that there are primarily seven different types of file outputs from different modules in the FMP. The most important feature of the FMP is that the standardization is introduced in four of these seven file types. They are:

#2 - IGES file

#3 - feature file

#5 - process plan file

#6 - M & G code file

A summary of different file types within the FMP is shown in table 2.1.

2.3.1 IGES File

A Summary of Files within the FMP

There are seven types of files used within the FMP. For some files, the file format is predecided. For the remaining files, there is some lattitude in the file format selection.

File description	Predecided	
	Yes	No
(1) Drawing files	x	
Geometry data (binary)		
(2) IGES files	x	
Geometry data (ascii)		
(3) Feature files		x
Geometry data		
Topology data		
Feature data		
Tolerances		
(4) Intermediate files		x
All of the above		
Fixturing		
(5) Process plan files	x	
All of the above		
Process Plans		
(6) M&G code files	x	
M&G codes for file (5)		
(7) NC code files	x	
NC code for file (6)		

Table 2.1 FMP files

IGES stands for the Initial Graphics Exchange Specification developed in late 1979 by the National Bureau of Standards (NBS). IGES is a mature mechanism for the digital exchange of database information among present day CAD systems. Engineering drawings, 3-D wireframe and surface part models, printed wiring product descriptions, finite element mesh descriptions and process instrumentation diagrams are addressed by the contents of Version 3.0 [43,44]. The IGES Version 3.0 specifies a file structure format, a language format, and the representation of geometric, topological, and non-geometric product definition data.

2.3.2 Feature File

Whereas IGES has addressed the need for data exchange where the received product model is interpreted by a human either as a display or as a generated plot, the Product Data Exchange Specification (PDES) project, begun in mid-1984 by the IGES Organization of NBS, is focused on exchanging product models with sufficient information content. PDES supports not only the geometry, but also a wide range of non-geometry data such as manufacturing features, tolerance specifications, material properties and surface finish specifications. Solid representations will also be included in the geometry model of PDES. It is the intent of the PDES to fully support the needs for a complete product model [43,45]. However, PDES is currently under development with the first version tentatively set to be released in mid 1988.

Another alternative to standardization is the Part Model Format (PMF). PMF is the format currently being used by NBS as the part model to transfer part information. It carries also the geometric, topological, feature, and manufacturing information. Unlike PDES, it deals only with machinable parts, and it supports only boundary representation solid modeler. At present, there is still some latitude in choosing the feature file format so that either PDES or PPF can be selected as the format to be attached to the FMP in the near future.

2.3.3 Process Plan File

The process plan file describes the manufacturing activities to be carried out at system level and cell level to accomplish the manufacturing tasks.

A survey shows that there is no standard in the process plan file format at present. However, the Process Plan Format (PPF) used by the AMRF (Automated Manufacturing Research Facility) of NBS has the potential to become the standard of tomorrow and is selected to be the process plan file format for the FMP. The process plan file format will be discussed in depth in later chapters.

2.3.4 M & G Code File

M & G Codes are the standard for cutting codes for all NC/CNC machines and are specified in EIA RS-274D. The G codes are known as the preparatory functions and define the actual machining moves and cutter tool movement. The M codes are known as miscellaneous function control codes such as the program stop, spindle ON/OFF, coolant ON/OFF, etc. In both M codes and G codes, there are some codes which are unassigned and left for the user to assign for special functions.

In addition to M & G codes, F function defines the feed rate in X, Y, Z axes, T function defines the tool and S function defines the surface speed of cutter.

2.4 Validation of FMP and Problem Formation

By taking a close look at the FMP (fig. 2.1) from the function's point of view, we can represent the FMP by two different blocks (fig. 2.2). The top block represents the FMP/D (Part Design) that prepares all the required manufacturing related data, such as feature file, process plan file, and NC code file for a particular part. The

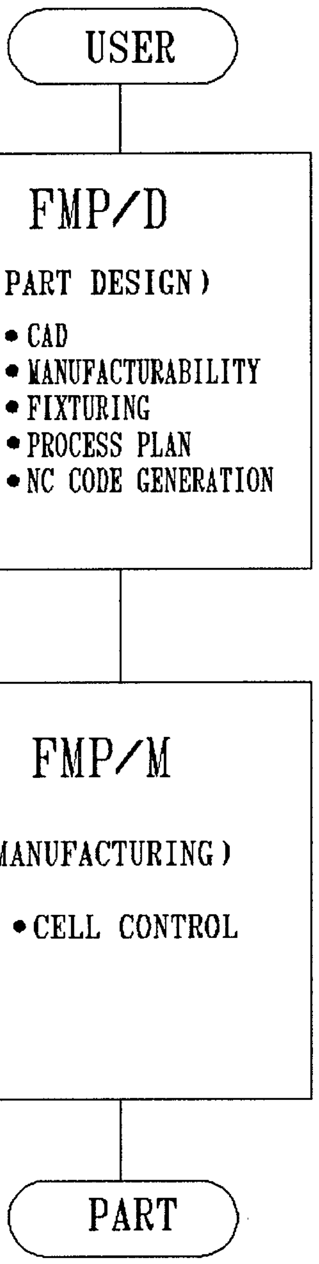


Figure 2.2 FMP/D and FMP/M

lower one is FMP/M (Manufacturing) where the manufacturing processes take place, such as transportation of workpiece, loading/unloading of the part, and machining.

The philosophies of the FMP/D have been partially validated by previous works [4,14,19,42,46,47], however, the link between the FMP/D and FMP/M as well as the inside of the FMP/M remains unexplored. The objective of this thesis is to identify the information required for this link and set up a vertical machining cell for automated machining to validate the FMP concept. Work involved during the development of the cell includes the identification and determination of cell environment, cell input/output (I/O), cell structure, control scheme, and information flow, development of cell level softwares, interfacing and implementation of software modules as well as hardware modules, and validation of cell capability.

CHAPTER 3

THE UMCP VERTICAL MACHINING CELL

3.1 The Cell Concept

To adequately meet the wide range of system applications, manufacturing systems have been divided into three types; special systems, flexible manufacturing systems, and flexible manufacturing cells. The degree of production flexibility is the major difference between the system types. The flexible manufacturing cells are the most flexible of the three types and viewed as being effective when applied to the production of many different workpieces with each being produced at a comparatively low production rate.

In order to achieve the high flexibility and make the implementation of the design of FMC tractable, two principles were suggested by the Robotics Institute of CMU [25]:

1. The cell and its component parts and pieces must be modular.
2. The cell and its components fit in a structured hierarchy.

The modular approach is a well-known principle in the design and production of various items and equipment. This idea is also most appropriately applied in the structural development of the computerized flexible production systems. This is the main requirement for staying competitive and the modular approach helps in making use of the advantages of unification and standardization to the maximum. It allows the user to distribute the investments over a long period of time, to gain experience, to refine the software and to proceed towards complete automation [20].

The hierarchical concept which consists of several control levels, as shown in fig. 3.1, has been generally accepted as the control structure of a flexible automation system [11]. It provides a method for partitioning the control problem into modules so that each module can be implemented as a finite-state automation [18]. Although supercomputer control (central control) can improve the performance of a system by its enhanced computational power and make some methodologies plausible which were prohibitively time consuming for a real time application with the conventional computers, the use of hierarchical control distributes the intelligence. Control systems based on distributed intelligence have less difficulty in responding to the external environment because it ensures that the size, functionality, and complexity of individual control modules is limited [24, 27].

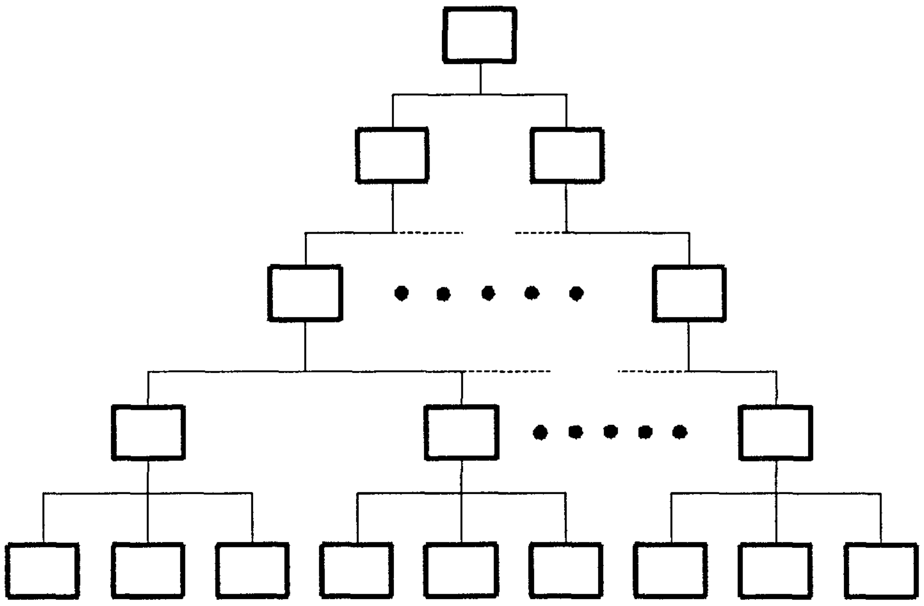


Figure 3.1 Typical hierarchical control structure

Information in the system must be accessible when and where it is needed. As the system grows, the need for a general management system and Local-Area Networks (LANs) become apparent for handling the large amount of information. LANs provide better communication at the local level, helping users assimilate vast amount of data from sensors, digital computers, and microprocessors. It consists of a number of computers or computer-assisted devices, electronic hardware interfaces, specialized software, and a cabling medium for interconnection. Ethernet is one of the popular medium that serves as a common cable to which the computers and their network interfaces connect [23].

3.2 Cell Configuration and System Hierarchy

It is the goal of the FMLAB to set up two machining cells, viz. the vertical machining cell and the turning cell. Each cell has its own FMP. This thesis addresses the design and implementation of the vertical machining cell (cell #1) and its associated FMP.

The cell configuration of FMLAB is defined as shown in fig. 3.2. The cell consists of a robot for workpiece transportation inside the cell and a machining center. Other components include a gripper and vise that work

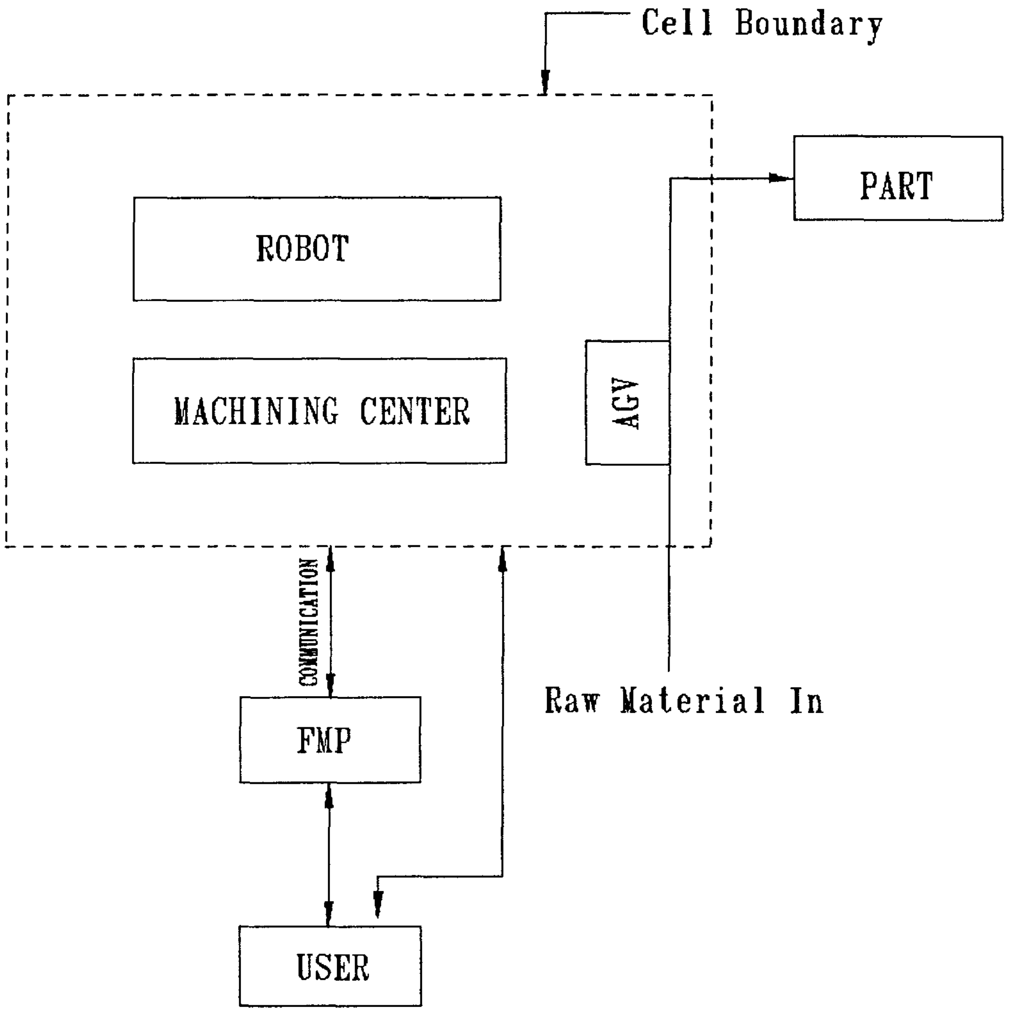


Figure 3.2 Cell configuration

together with the robot arm and machining center, respectively. The dashed line represents the cell boundary with the AGV traveling through the cell boundary and delivering the workpiece. The user accesses the cell facility either by the direct link or by using the FMP where the user can design the part and prepare the manufacturing related data. In order to view the role of the machining cell in the FMP, fig. 2.2 is re-emphasized as shown in fig. 3.3.

Based on the concept of hierarchical control and modular approach, we have defined the system structure as shown in fig. 3.4. At the top level of the hierarchy is a system computer that manages the cell level components, distributes and schedules the production tasks. Under the system computer level are the cells. An equipment controller is put between the cell #1 host and the shop floor components to access the intermediate operation information of the cell. Notice that the AGV control module is at the same level as cell host in order to serve as a material transportation cell delivering parts between cells. The relationship between parent and child modules is master-slave relationship. No direct handshaking is allowed between any two modules at the same level of hierarchy. Each module in each level responds to its parent module and child modules only. The relationship between parent and child modules is master-slave relationship, and the

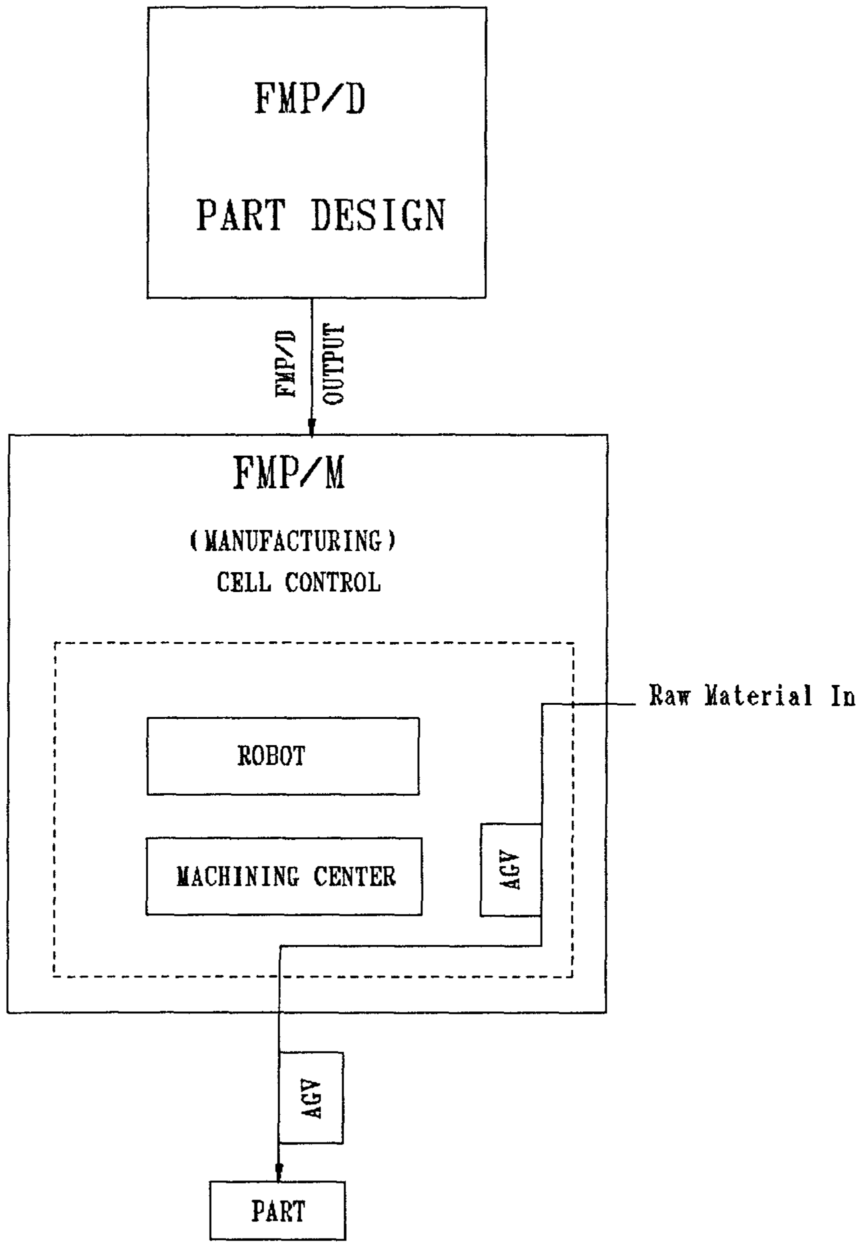


Figure 3.3 FMP and cell

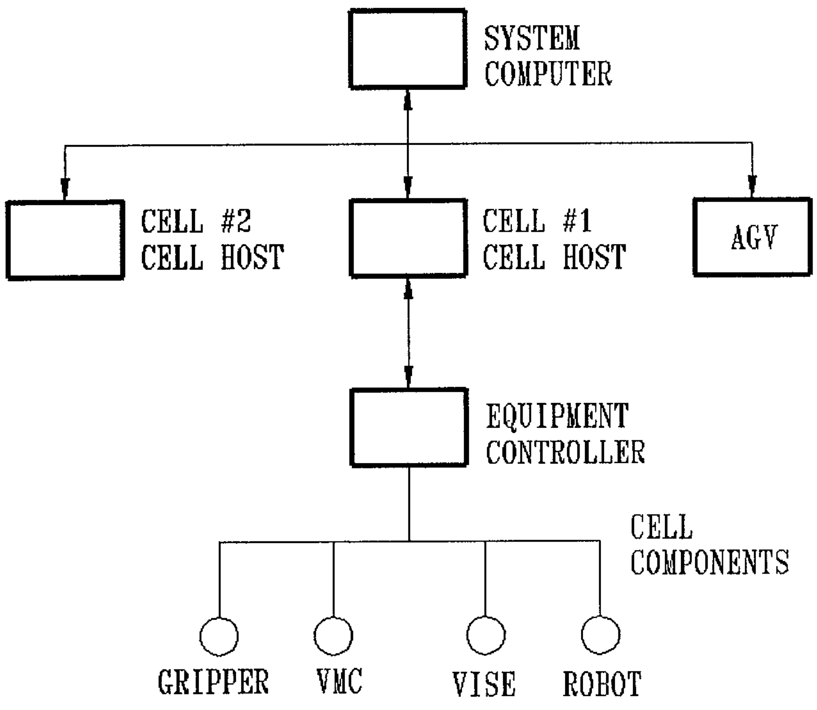


Figure 3.4 FMP system structure

information path between modules at the same level is controlled and handled by parent module or modules at a higher level.

3.3 Cell Capability and User Environment

The vertical machining cell is designed for automated machining with the following capabilities:

- The cell will handle prismatic blanks.
- Manufacturing using the FMP.
- Tools limited to the tool magazine.
- The cell will be able to machine without fixture intercepts.
- Equipment status is always known.
- One part for each AGV run.

The FMP system is designed to be a multi-user system. There are three possible ways that the user can access the FMP facilities (fig. 3.5):

1. Access the FMP from the system workstation or console (system context).
2. Access the FMP from a remote PC as a terminal (system context).
3. Access the FMP from a remote PC as the computer (remote computer context).

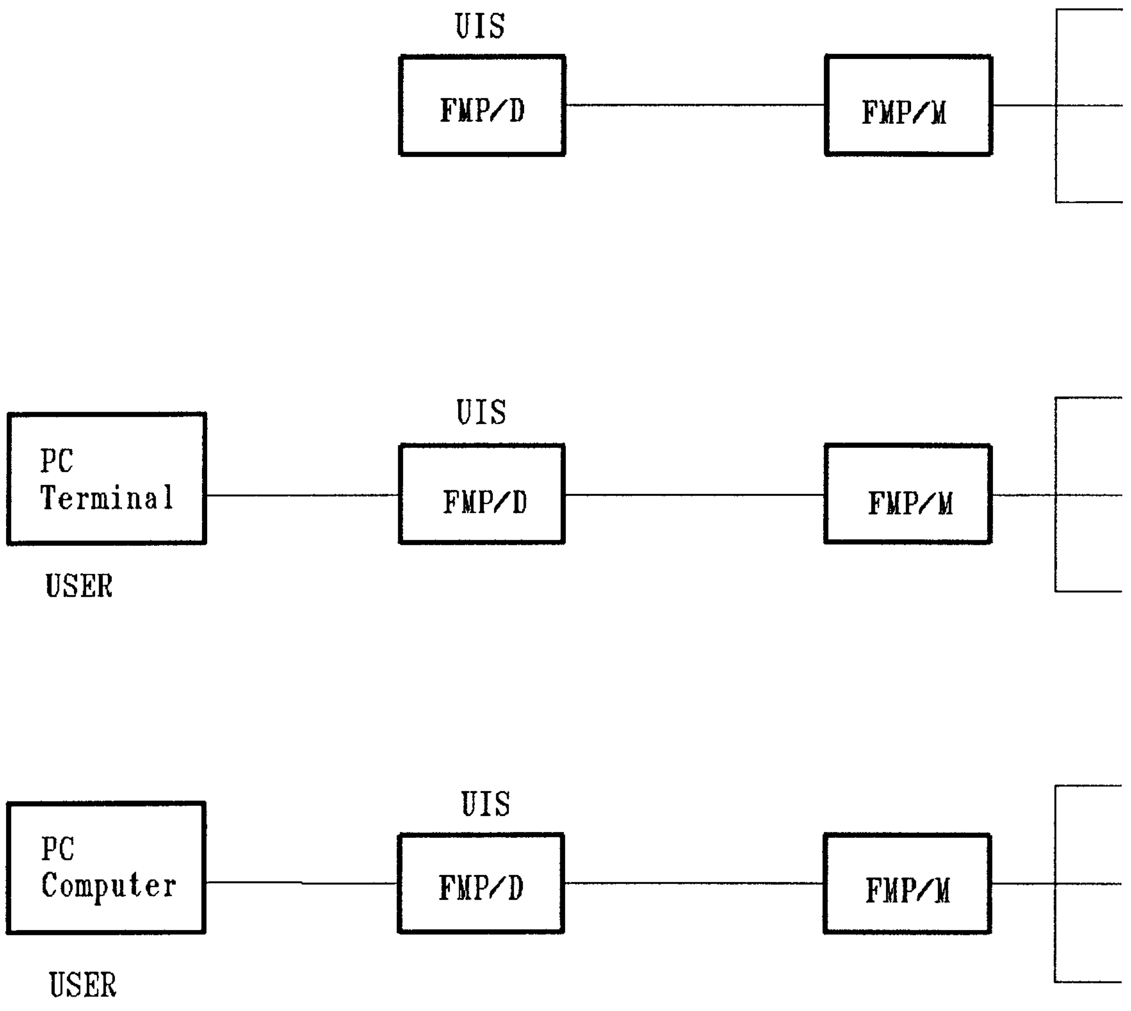


Figure 3.5 Three possible ways of access FMP

However, method 3 requires dedicated software, which is machine-dependent, running in the remote computers to interact with the FMP and also complicated network problems are involved. It is therefor inappropriate to use method 3 in the present application.

For each level of cell hierarchy, there will be manual mode, automode, and help mode. With the manual mode, the user will be able to access the status of components in the next level of hierarchy as well as control these components. When in automode, the control is then returned to the upper hierarchy. These modes will help the user in setting up the cell, controlling individual component, diagnosing the cell, and recovery from failures.

3.4 Constraints

Due to the fact that the intelligent vise, gripper, and AGV are not available at the current stage of development of the cell, the control of these components is simulated by several personal computers together with human intervention. The personal computer takes in the command from the parent module of the simulated component and shows it on the screen. The human reads the command from the screen, executes it, and responds to the PC with the current status of the component or the status of execution. The PC then converts the status into a specific format and returns the information to the commanding module.

This simulation strategy is very important in that it provides a dynamic model to simulate all possible combinations of responses from different controllers. It assures that the present step-by-step integration of the cell will lead to a fully automated cell.

CHAPTER 4

CELL CONTROL

4.1 Software Module of The Cell

In order to define the software modules that are needed, a clear understanding of the types of functions that are to be accomplished is necessary. Table 4.1 briefly describes the functions to be accomplished by this software.

By mapping table 4.1 onto the proposed system hierarchy (fig. 3.4), the software structures are defined as shown in fig. 4.1. Functions performed by each module are described in the following paragraphs.

User Interface Program (UIP) module - This is the program through which the user will access the utilities of the system. More than one user will be allowed to log in and run this program. UIP provides optional entries in the FMP/D for the user to select from. It is also the program through which the user can enter the System Control Program (SCP) of FMP/M in order to carry out the manufacturing tasks. Fig. 4.2 shows how the UIP Shell (UIS) encompasses the FMP/D and FMP/M.

System Level	- to interface the user and system. - to take in user's request to produce a part, coordinate and control the activities at system level. - to simulate AGV system.
Cell Level	- to simulate gripper, vise, robot and the machining center. - to prepare cell level commands. - to coordinate the activities of cell components. - to carry out the cell level commands.

Table 4.1 Functions to be accomplished

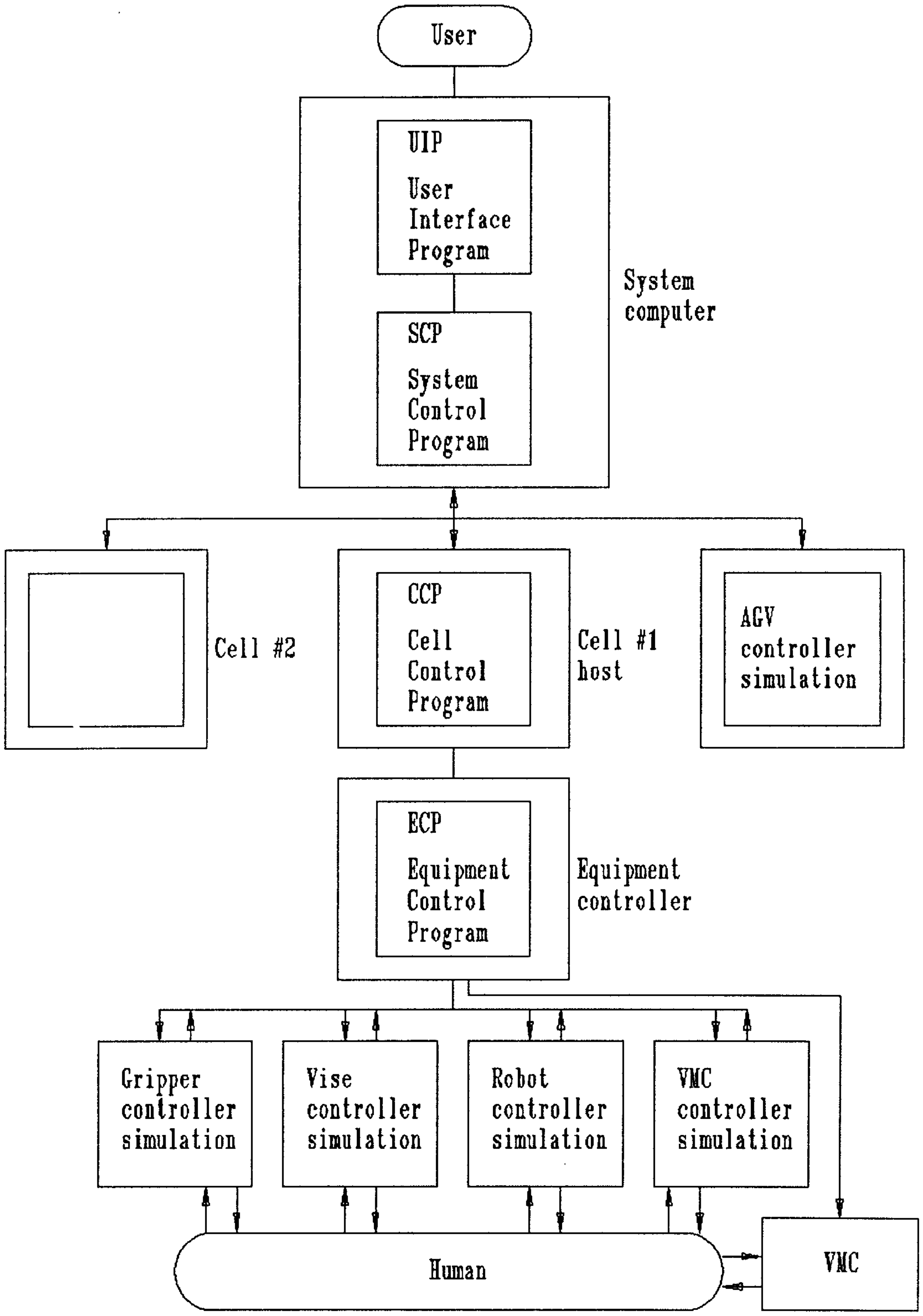

User
UIP
User Interface Program
SCP
System Control Program
System computer
Cell #2
CCP
Cell Control Program
Cell #1 host
AGV controller simulation
ECP
Equipment Control Program
Equipment controller
Gripper controller simulation
Vise controller simulation
Robot controller simulation
VMC controller simulation
Human
VMC

Figure 4.1 FMP software structure

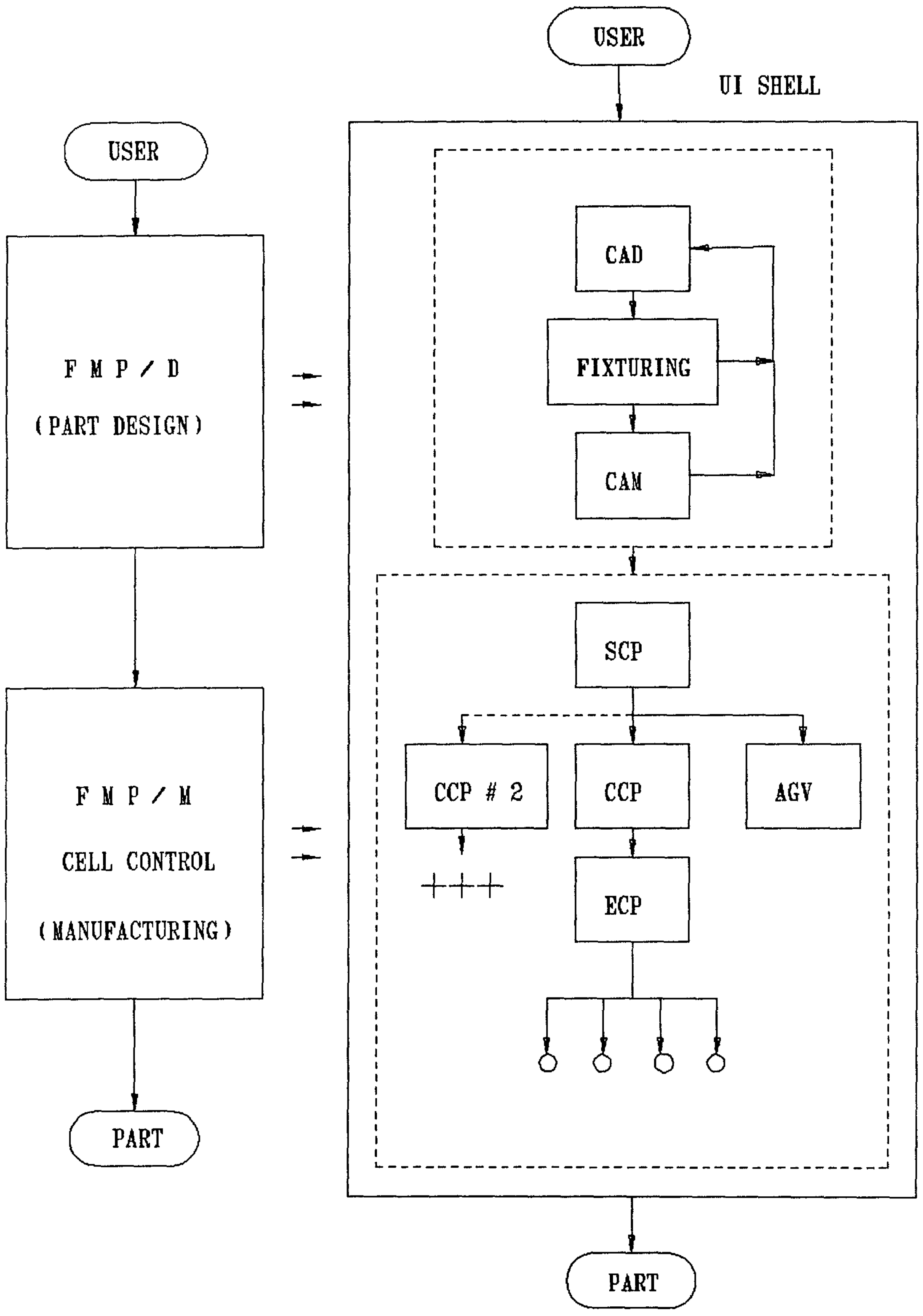


Figure 4.2 UIS and FMP

System Control Program (SCP) module - This is the program that manages, controls, and monitors all of the cells. It will accept system level process plans as input in the automatic mode. It will also accept system level work elements (ex. command AGV to go to a target location) in manual mode. Only one SCP can run at a time.

Cell Control Program (CCP) module - This is the cell host of the vertical machining cell. It will accept and execute the cell level process plan. CCP first decomposes the process plan into a collection of cell level commands, relay these commands one at a time to the equipment controller and assures that the status of the command is in the correct status. It also accepts cell level command in the manual mode.

Equipment Control Program (ECP) module - Its major responsibility is to accept cell level commands issued from cell host and execute those commands on the appropriate piece of equipment.

Cell component simulation programs - it is assumed at some point that there will be intelligent programs running on the robot, gripper, vise, and VMC, respectively, which will have the capability to accept commands from ECP, execute the command, and return the status. This will be simulated in the present test by personal computers. These

PCs will accept commands, wait for human intervention (ex. loading part, keyboard input) and then return the status message based on the input of the user at the keyboard.

All the software modules inside the cell boundary have been developed and implemented. Interfaces of each module are under development and are discussed in later chapters.

4.2 Hardware Module of The Cell

The hardware available for the vertical machining cell implementation includes a Matsuura 510 CNC milling center, a SUN 3/160 workstation with 16 MB of RAM and 280 MB of secondary memory, a HP 310 bundle system in BASIC Operation System, and several IBM ATs. Fig. 4.3 shows how these equipments were mapped into the control hierarchy and software structure. The gripper and vise perform similar functions and hence are simulated by the same computer. Similarly, the robot and VMC perform coordinate positioning functions and hence their simulation programs reside in the same computer.

4.3 FMP/D - FMP/M Interface

In this section the software interface between FMP/D and FMP/M (fig. 4.2) is discussed. The SCP needs a driver which defines the sequence and procedures to be carried out to accomplish the manufacturing task. This includes the

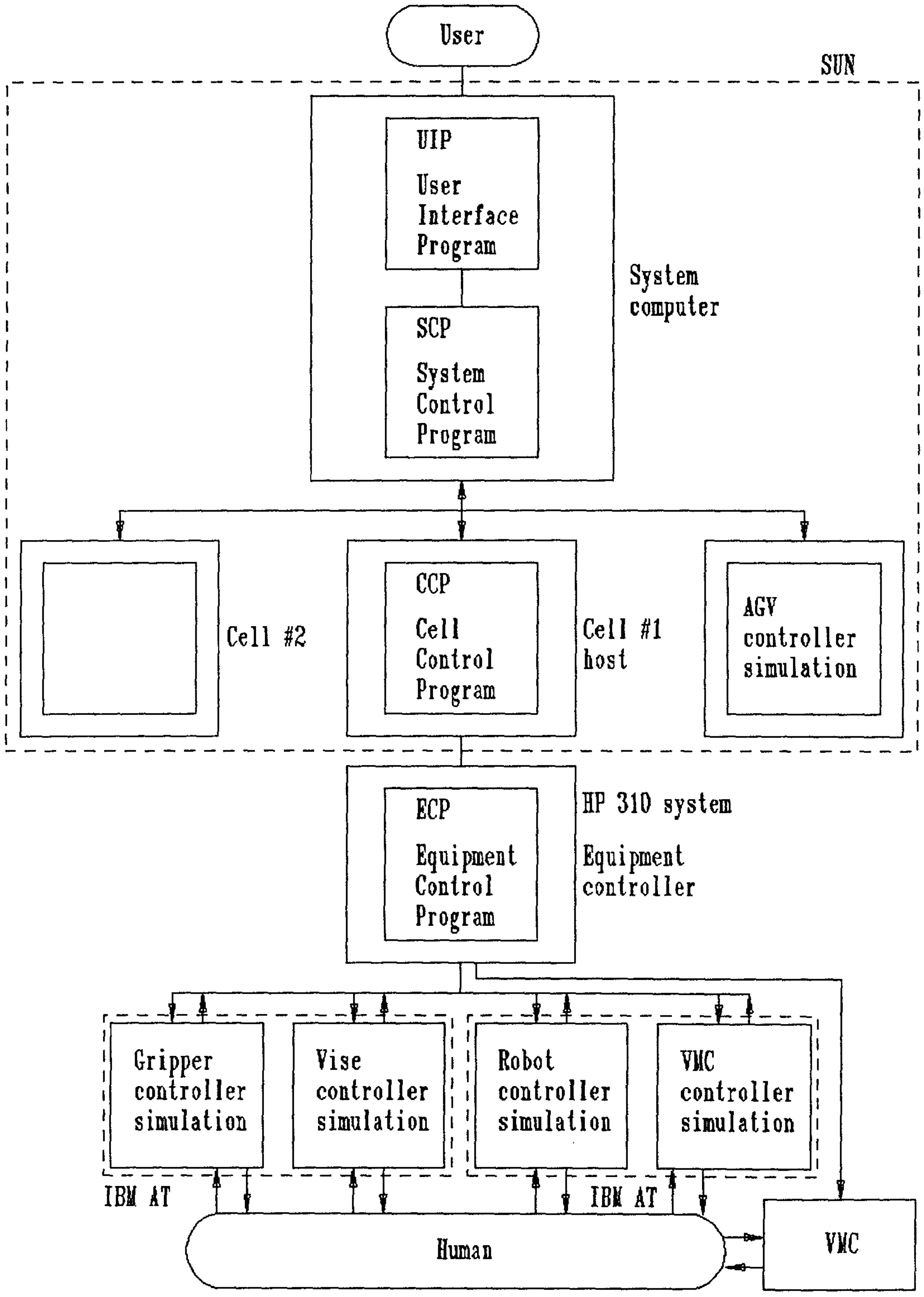


Figure 4.3 FMP hardware map

control of system level components, such as control of AGV to travel through cell boundary and control of cell components, such as the vise, gripper, robot, and VMC, which can not be seen by SCP. The SCP also needs the NC codes for the specific machining tool to cut the part. The process plan file in FMP (fig 2.1, table 2.1, fig 2.2) should provide the former information while the postprocessor provides the NC code file. The information required for the SCP is thus defined as the process plan files and the NC code file.

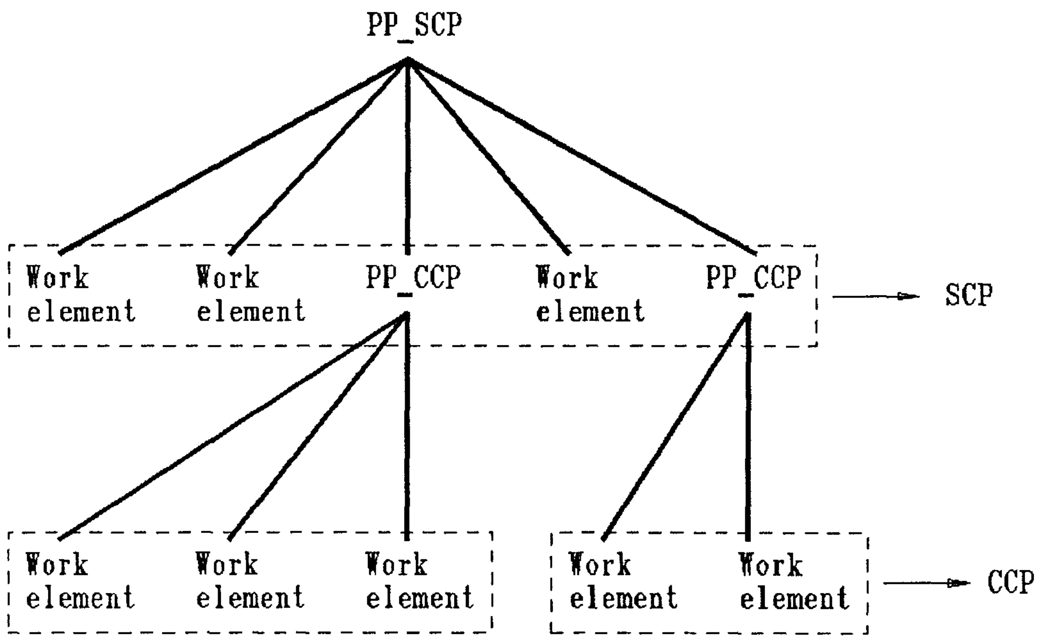
There are two types of process plan files: the system level process plan files (PP_SCP), which are used to drive the SCP, and the cell level process plan files (PP_CCP), which are used to drive the cell host (CCP). Each manufacturing task of a particular design consists of a SCP process plan file and one or more CCP process plan files generated by the processor planner in FMP/D. The relationship between PP_SCP and PP_CCP is parent-child relationship as shown in fig. 4.4.

The work element is defined as the lowest level of manufacturing commands that are recognized by the process planner. PP_SCP consists of several work elements and may also refer to the PP_CCP which consists of only work elements. Fig. 4.5 shows the work elements currently available in PP_SCP and PP_CCP.

Process Plan for FMP:

. Process plan for SCP level

. Process plan(s) for CCP level



* Each work element will be decomposed into a set of cell level commands by CCP, NC code filename may be passed to CCP by the work element as an argument.

Figure 4.4 PP_SCP and PP_CCP

Work Elements at SCP level :

. INIT

. CLOSE

. AGV_TRAN
  . origin
  . destination
  . type of contents

. PP_CCP
  . host name
  . plan ID
  . plan version
  . plan type
  . plan name

Work Elements at CCP level of cell#1 :

. INIT

. CLOSE

. LOAD_PART
  . part size
  . part type
  . part material

. UNLOAD_PART
  . part size
  . part type
  . part material

. MACHINE_PART
  . NC file name

Figure 4.5 Work elements

The general structure of a process plan file consists of 4 different sections: the header section, the parameters section, the requirements section, and the procedure section. The header section defines the general information about the file such as the plan ID, plan version, plan type, and plan name. The parameters section declares undefined parameters which will be defined at run time. The requirements section declares required child process plan files and hardware components. The procedure section describes the procedure for accomplishing the given task.

Shown in fig. 4.6 is an example of the PP_SCP process plan file called PART01_01. The requirements section shows a child process plan (PART01_01_1) that will be executed and the required facilities for PART01_01 including CELL_1 (cell #1) and AGV_1. The procedure section shows the procedure of the manufacturing events: the AGV transfers a part from LOT to CELL_1, the cell host of CELL_1 executes process plan PART01_01_1 to machine the part "DEMO_PART" and, after the part is machined, the AGV comes in CELL_1 and transfer the machined part to LOT. Appendix A provides additional details on the PPF format.

4.4 Control Strategies

General Structure of a Process Plan File

. HEADER SECTION - general information about the file.

. PARAMETERS SECTION - declares undefined parameters which will be defined at run time.

. REQUIREMENTS SECTION - declares required child process plan files and hardware components.

. PROCEDURE SECTION - describes the manufacturing processes in sequence

Figure 4.6(a) Process plan file structure

PP_SCP

```
-- PROCESS_PLAN --


-- HEADER_SECTION --
    PLAN_ID         := PART01_01;
    PLAN_VERSION    := 1;
    PLAN_TYPE       := MILLING PART;
    PLAN_NAME       := DEMO_PART;
-- END_HEADER_SECTION --


-- PARAMETERS_SECTION --
-- END_PARAMETERS_SECTION --


-- REQUIREMENTS_SECTION --
<<1>> PROCESS_PLAN
    (PLAN_ID              => PART01_01_1,
     PLAN_VERSION         => 1,
     PLAN_TYPE            => MILLING PART,
     PLAN_NAME            => DEMO_PART );


<<2>> CELL
    (CELL_ID              => CELL_1 );


<<3>> AGV
    (AGV_ID               => AGV_1 );
-- END_REQUIREMENTS_SECTION --
```

Figure 4.6(b) Example of a PP_SCP file (PP. 1 of 3)

```
-- PROCEDURE_SECTION --
<<1>> INIT


<<2>> AGV_TRAN
    (ORIGIN              => LOT    ,
     DESTINATION         => CELL_1,
     CONTENTS            => PART   ,
     PREC_STEPS          => ()     ,
      TIME                => 0000:00:01:00);


<<3>> PP_CCP
    (HOST_ID             => CELL_1        ,
     PLAN_ID             => PART01_01_1   ,
     PLAN_VERSION        => 1             ,
     PLAN_TYPE           => MILLING_PART  ,
     PLAN_NAME           => DEMO_PART     ,
     PREC_STEPS          => (2)           ,
     TIME                => 0000:00:10:00 );


<<4>> AGV_TRAN
    (ORIGIN              => CELL_1   ,
     DESTINATION         => LOT           ,
     CONTENTS            => PART          ,
     PREC_STEPS          => (3)      ,
      TIME                => 0000:00:01:00 );
```

Figure 4.6(b) Example of a PP_SCP file (PP. 2 of 3)

```
<<5>> CLOSE

-- END_PROCEDURE_SECTION --


-- END_PROCESS_PLAN --
```

Figure 4.6(b) Example of a PP_SCP file (PP. 3 of 3)

In the previous sections, we have described the software modules of the control hierarchy, functions to be performed at each module, and the required information for the link between FMP/D and FMP/M. In this section, we will describe how the FMP works with the cell.

Typically the user sits at the top of the control hierarchy (fig. 4.3). From there he enters the UIP and then selects to enter the FMP/D to create a design, check the manufacturability of the design, do the process planning and generate the process plan files and NC code file. The user then can exit the FMP/D and, while still in UIP, enter the SCP of FMP/M to start the manufacturing of the design.

Based on the process plan filename given by the user, the SCP first executes the PP_SCP process plan and starts the manufacturing cycle. To explain the idea clearly, we assume that the PP_SCP process plan filename is PART01_01 (fig. 4.6). According to this process plan, the SCP commands the AGV to deliver the blank into cell #1 and, when completed successfully, issues the child process plan filename (PART01_01_1) to the cell host of cell #1. The NC code filename required for machining is also passed to the cell host implicitly by a parameter of one of the work elements in the child process plan.

The cell host takes in the child process plan PART01_01_1, decomposes the work elements in the procedure section one-by-one into cell level commands and, eventually, generates the Cell Command File (CCFILE) which is a collection of cell level commands. The cell host then relays these cell level commands, sequentially, to the equipment controller (ECP) which will execute the received command on an appropriate component such as vise or gripper. After the received command is executed, the ECP signals the cell host (CCP) and waits for the next incoming command from CCP. Once the last command in CCFILE is executed, the task of the cell is completed. The cell host reports the status of execution of child process plan PART01_01_1 to SCP. According to PART01_01, the SCP then commands the AGV to deliver the machined part to storage and terminate the manufacturing cycle.

CHAPTER 5

SOFTWARE INTERFACES

5.1 Module Input/Output

Input/Output (I/O) of each module in FMP/M is classified according to three major types of information: request type, status type, and data type. The request type of information can be further decomposed into two sub-types: service request type and status request type. Service request type encompasses all the requests that require services from the hardware component, such as vise, robot, and gripper. Status request type includes all the requests for status of a particular hardware component or a group of hardware components. Status type of information covers all the status reports as the result of status requests. Data type includes all the files that are to be used in the manufacturing processes such as PP_SCP, PP_CCP, CCFILE, and NC code file.

Request type of information must go down to the module in the next lower hierarchy or come in from the module in the next upper hierarchy. Status type of information must come in from the module in the next lower hierarchy or go out to the module at the next higher hierarchy. This type of classification assures that no handshaking will occur between any two neighboring modules at the same level of hierarchy. With this classification, together with the information given in chap. 4, we thus can define the Input/Output of each software module to achieve the required modularity.

Shown in fig. 5.1 are the Input/Output diagrams of the software modules in FMP/M. Since the robot, vise, gripper, and VMC are at the lowest level of the hierarchy, the input/output of each of these modules must form a closed loop. The input must be from the equipment controller, and the output must be fed back to equipment controller. Similarly, the I/O of the ECP can be connected to CCP, and so on. Fig 5.2 shows how all the modules are connected together. Notice that the communication between CCP and SCP is done through the use of a mailbox. This increases the flexibility of the cell so that in case the SCP is idle or down, the cell utility is still available for some other system.

5.2 Input/Output Format

Notice that although the PP_SCP is the driver of SCP, it is the service request from the user that activates the SCP to execute the PP_SCP. Similarly, although the PP_CCP is the driver of CCP, the service request from SCP activates the PP_CCP process. Thus, all software modules

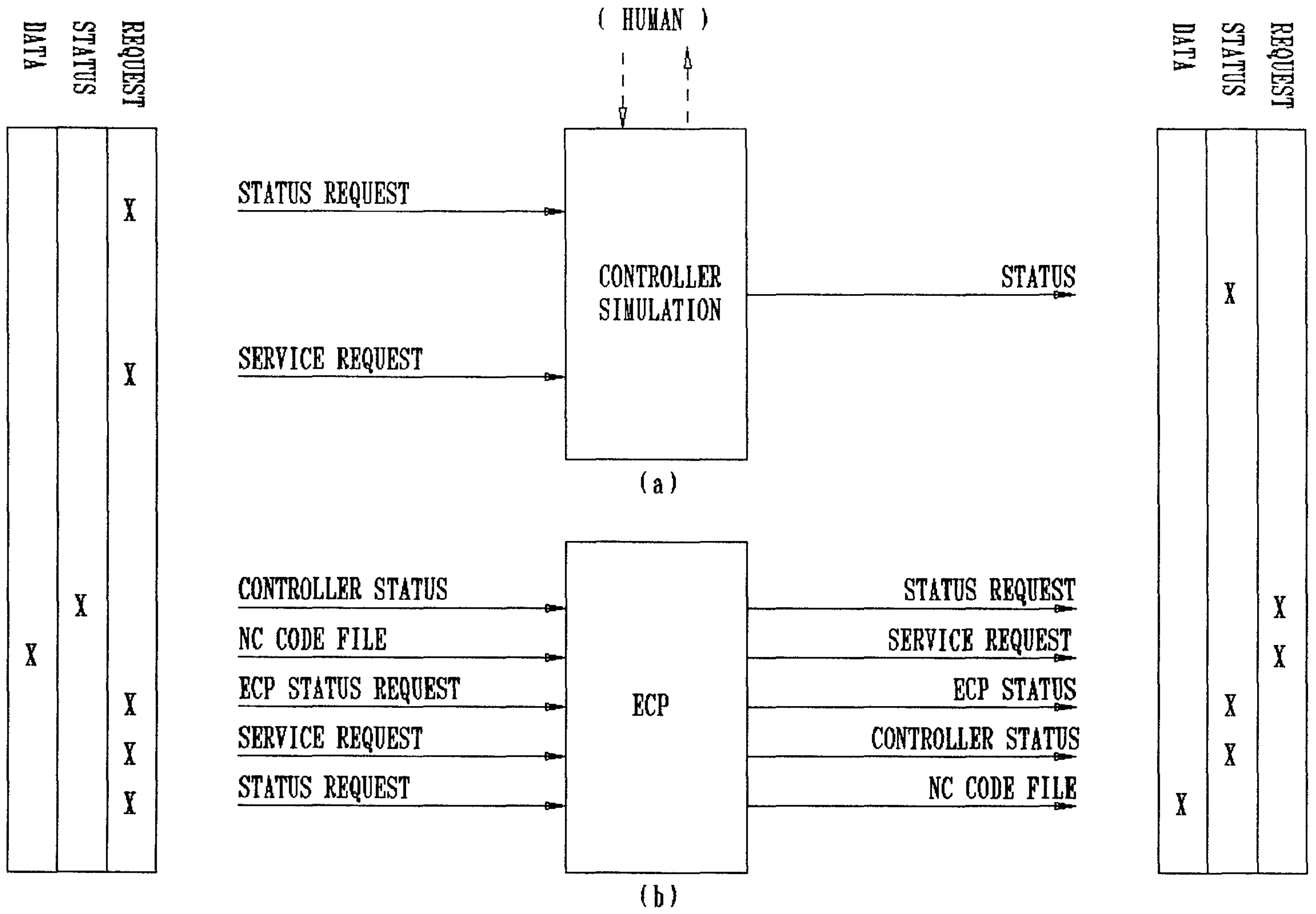


Figure 5.1 (a)(b) I/O diagrams of FMP software modules

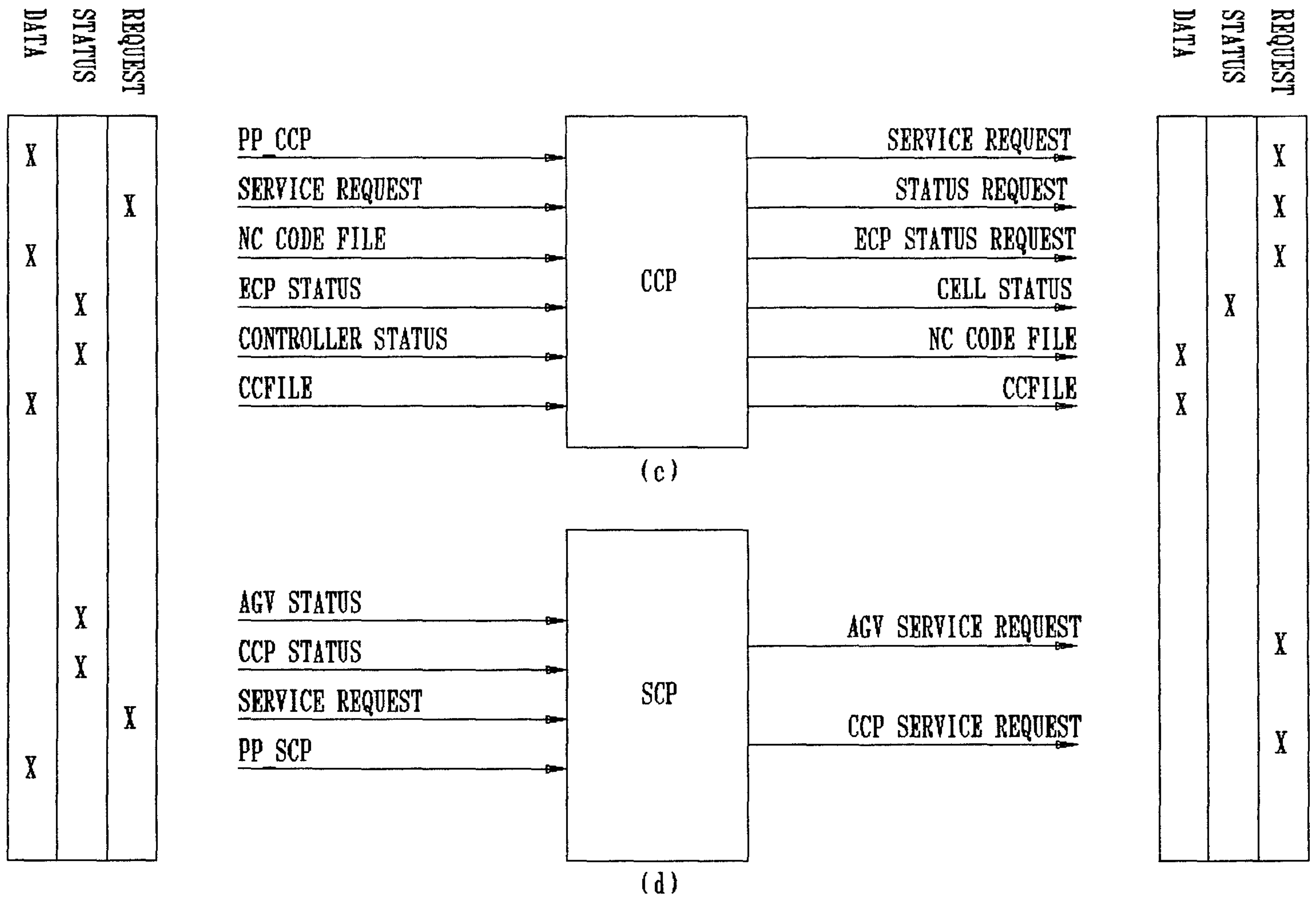


Figure 5.1 (c)(d) I/O diagrams of FMP software modules

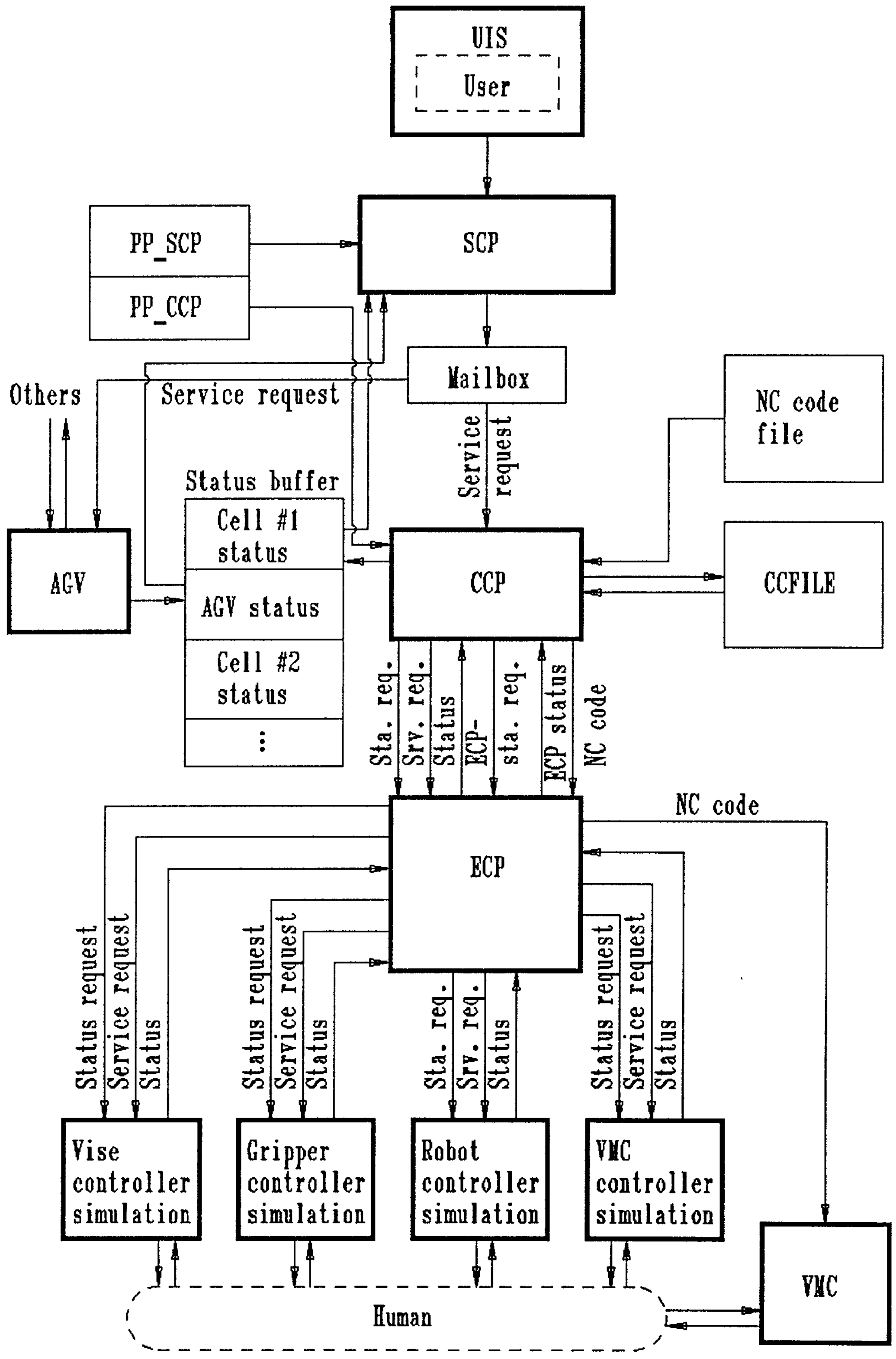


Figure 5.2 FMP software module links

are activated by such request type of information. Because this is a closed loop system, every request type of I/O will be associated with a status type of I/O. To simplify the complexity and increase the flexibility of the system, the I/O format is unified except for data type of information. Each data type of information has its own natural format that can not be changed. The unified I/O format will cover the request type (service request and status request) and the status type of I/O, cover all kinds of objects from the SCP to vise, and most importantly, allow the user to add parameters into the current format to increase the amount of information it carries. This I/O format will allow the system to adapt to changes in environment.

The fundamental I/O format consists of 64 characters (fig. 5.3) in one string. The first three digits (digit 0 to 2) is the field of information type such as status request type (STA), service request type (COM), or status type (STA) of information. Digit 4 to 11 (8 digits) is the target field, such as VMC, VISE, GRIPPER, ECP, SCP, etc. Digit 12 to 17 (6 digits) is the status field, such as READY, DOWN, BUSY, OFF, etc. Digit 18 to 62 (45 digits) is open field for parameters to be transferred. Digit 63 (the 64th digit) is the last digit and always contains "%" to indicate the end-of-message.

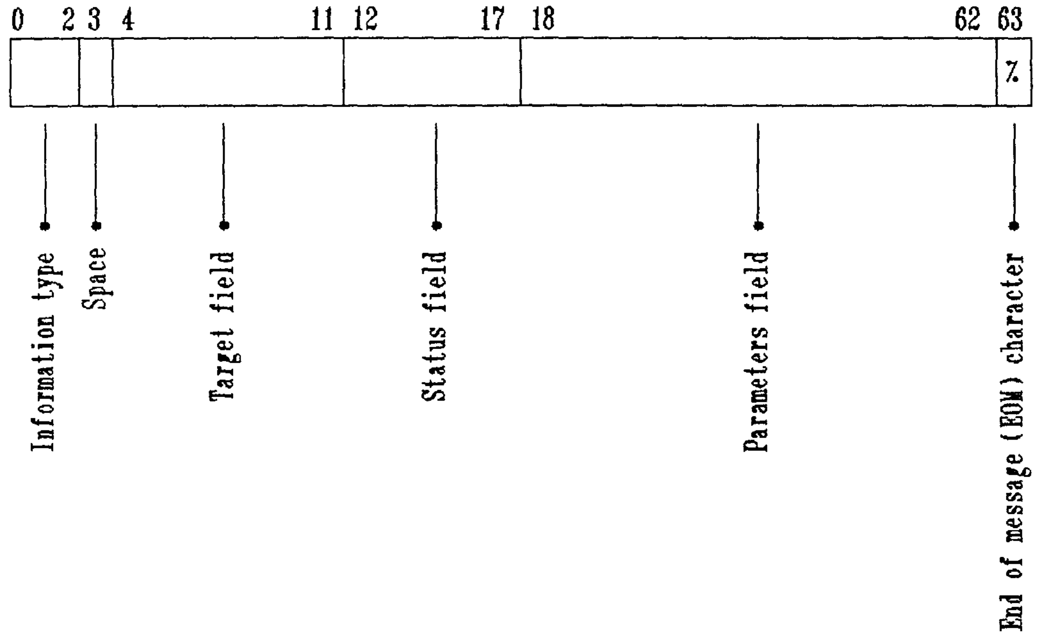


Figure 5.3 Fundamental I/O format

Based on this fundamental format, we first define the cell level command format. Recall that the cell level commands are generated by decomposing the work elements in PP_CCP (fig. 4.4). Every work element in PP_CCP is a task to be completed by one or more cell components. This implies that the target field can be filled in with one of the four components: VISE, GRIPPER, ROBOT, and VMC. The parameter field is divided into 3 sub-fields with 15 digits for each sub-field (fig. 5.4). The parameter can be the contents of the command, numerical values that shows the target point to be reached, or just blanks. Fig. 5.4 shows the cell command (request type of information) format for different objects.

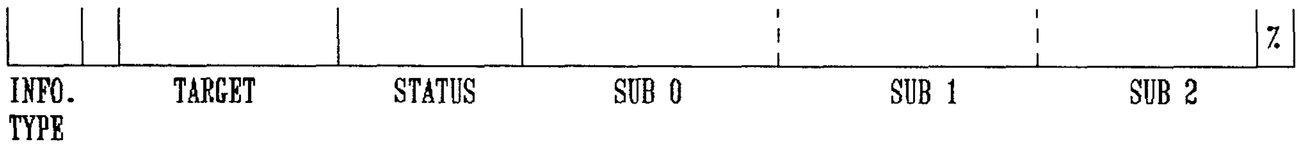
Another type of information that needs to be formated in the lowest level of hierarchy is the status type. Similarly, the format described previously for cell level commands can be applied to the status type of information. It has 3 parameters' sub-field, a target field, status field, and an information type (STA) field. Status field is one of the following: READY, BUSY, DOWN, and OFF. The sub-field reveals more information about the current status of the target. Fig. 5.5 shows the format of status type information in the cell level.

For the link between CCP and ECP, the format for request type and status type are exactly the same as those

INFO. TYPE	TARGET	STATUS	SUB 0	SUB 1	SUB 2	%

INFORMATION TYPE	COM				STA			
TARGET	VISE	GRIPPER	VMC	ROBOT	VISE	GRIPPER	VMC	ROBOT
STATUS								
SUB 0	OPEN CLOSE INITIAL	OPEN CLOSE INITIAL	INITIAL PRE_SET EXECUTE DOWNLOAD Num. value	INITIAL Num. value				
SUB 1			Num. value	Num. value				
SUB 2			Num. value	Num. value				
EOM	%	%	%	%	%	%	%	%

Figure 5.4 Request type format, ECP $\rightleftarrows$ controllers



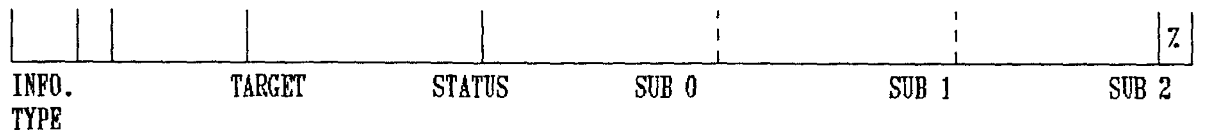
INFORMATION TYPE	STA			
TARGET	VISE	GRIPPER	VMC	ROBOT
STATUS	READY BUSY DOWN OFF	READY BUSY DOWN OFF	READY BUSY DOWN OFF	READY BUSY DOWN OFF
SUB 0	OPEN CLOSE	OPEN CLOSE	PRE_SET EXECUTE DOWNLOAD Num. value	Num. value
SUB 1			 File name Num. value	Num. value
SUB 2			 Num. value	Num. value
EOM	%	%	%	%

Figure 5.5 Status type format, controllers ⟶ ECP

at the lowest level, respectively, except there is one more target (ECP) for both. Fig. 5.6 and fig. 5.7 shows the format for both types of information running between CCP and ECP.

The format for the link between SCP and CCP is also similar to that discussed previously and is shown in fig. 5.8. From fig. 5.4 through fig. 5.8 we can find many open sub-fields in the request type command format. These blanks provide space for additional more information. Also, the parameter field can be rearranged to carry a maximum amount of information for some particular applications.

Fig. 5.8 shows that the SCP sends only manufacturing request to CCP rather than requests for service from a particular cell component. This is to ensure safety since most of the time a user working on SCP is actually "blind" to the conditions on the shop floor.



INFORMATION TYPE	COM					STA				
TARGET	ECP	VISE	GRIPPER	VMC	ROBOT	ECP	VISE	GRIPPER	VMC	ROBOT
STATUS										
SUB 0	INITIAL	INITIAL OPEN CLOSE	INITIAL OPEN CLOSE	INITIAL PRE_SET EXECUTE DOWNLOAD Num. value	INITIAL Num. value					
SUB 1				Num. value	Num. value					
SUB 2				Num. value	Num. value					
EOM	%	%	%	%	%	%	%	%	%	%

Figure 5.6 Request type format, CCP $\longrightarrow$ ECP

INFO. TYPE	TARGET	STATUS	SUB 0	SUB 1	SUB 2	%

INFORMATION TYPE	STA				
TARGET	VISE	GRIPPER	VMC	ROBOT	ECP
STATUS	READY BUSY DOWN OFF	READY BUSY DOWN OFF	READY BUSY DOWN OFF	READY BUSY DOWN OFF	READY DOWN
SUB 0	OPEN CLOSE	OPEN CLOSE	PRE_SET EXECUTE DOWNLOAD Num. value	Numerical value	
SUB 1			File name Num. value	Numerical value	
SUB 2			Num. value	Numerical value	
EOM	%	%	%	%	%

Figure 5.7 Status type information, ECP $\longrightarrow$ CCP

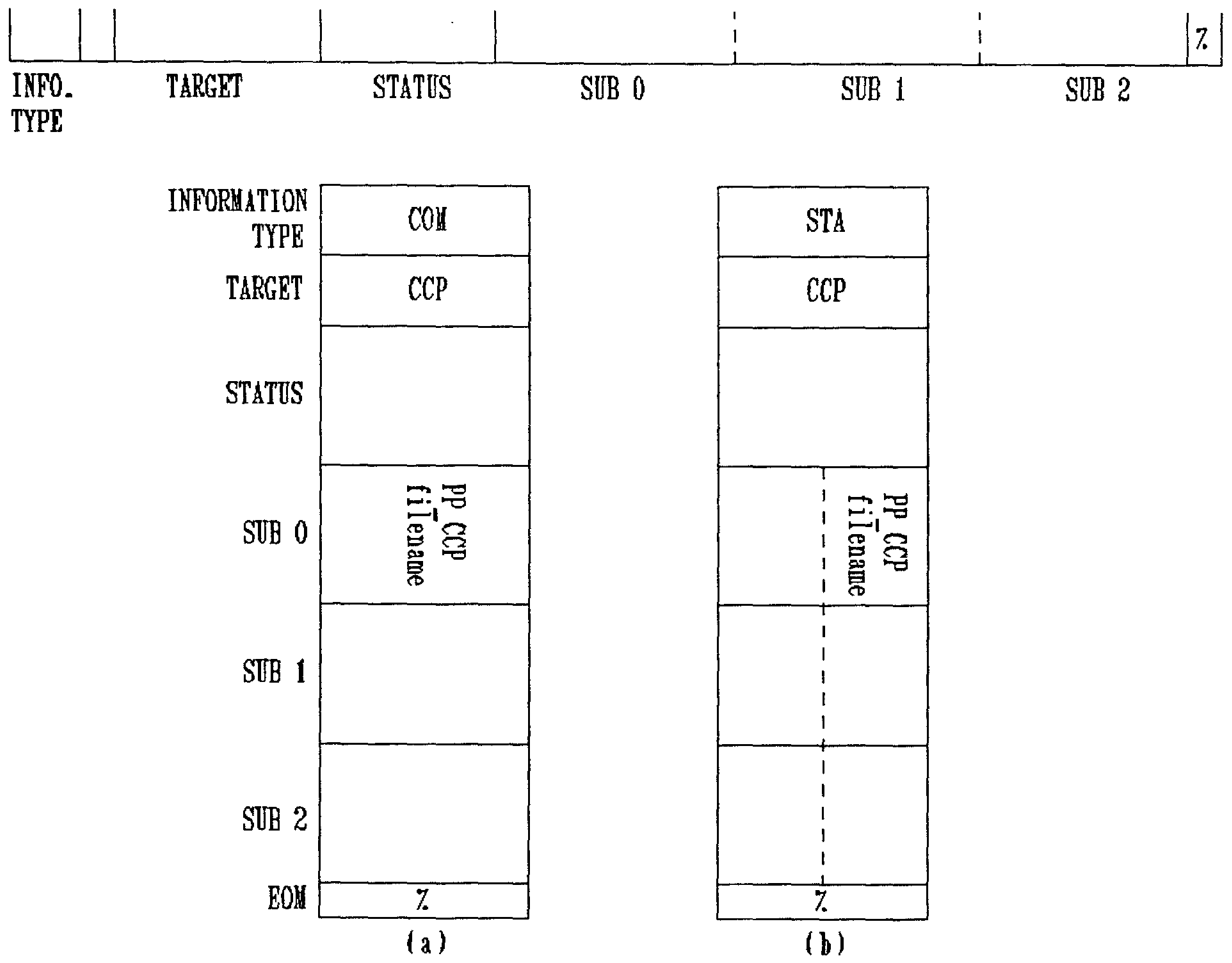


Figure 5.8 (a) Request type format, SCP $\rightarrow$ CCP
(b) Status type format, CCP $\rightarrow$ SCP

CHAPTER 6

SOFTWARE DEVELOPMENT AND IMPLEMENTATION

6.1 Flowchart of Each Level

In this section the functional flowcharts at each level of hierarchy are presented in order to show the sequence of functional operations and the environment that the system provides to users.

Shown in fig. 6.1 is the UIP flowchart which provides the user with the UIS (User Interface Shell). In this shell, the user will be able to access the UIS utilities such as file management, time sheets, and on-line help. For example, the user can manage the FMP-related files such as the design files, process plan files, etc. It is also the program through which the user is able to select the entry point of FMP. The user can select to either enter the FMP/D directly or go to the FMP/M by entering the SCP.

Fig. 6.2 shows how SCP works for the user. It first looks at the process plan files and NC code file to check whether the information required to produce the particular design specified by the user is correctly prepared. If all the data is prepared, it then checks the status of all the components mentioned in the requirements section of the

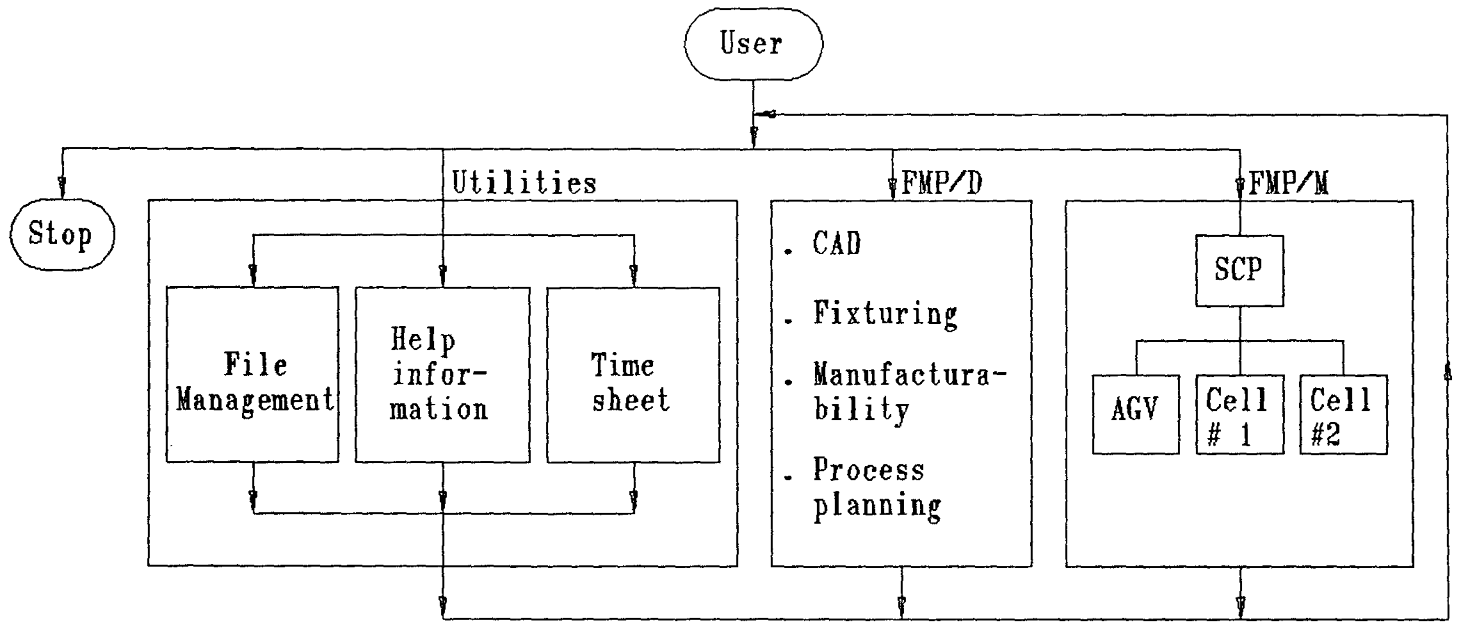


Figure 6.1 User Interface Shell (UIS)

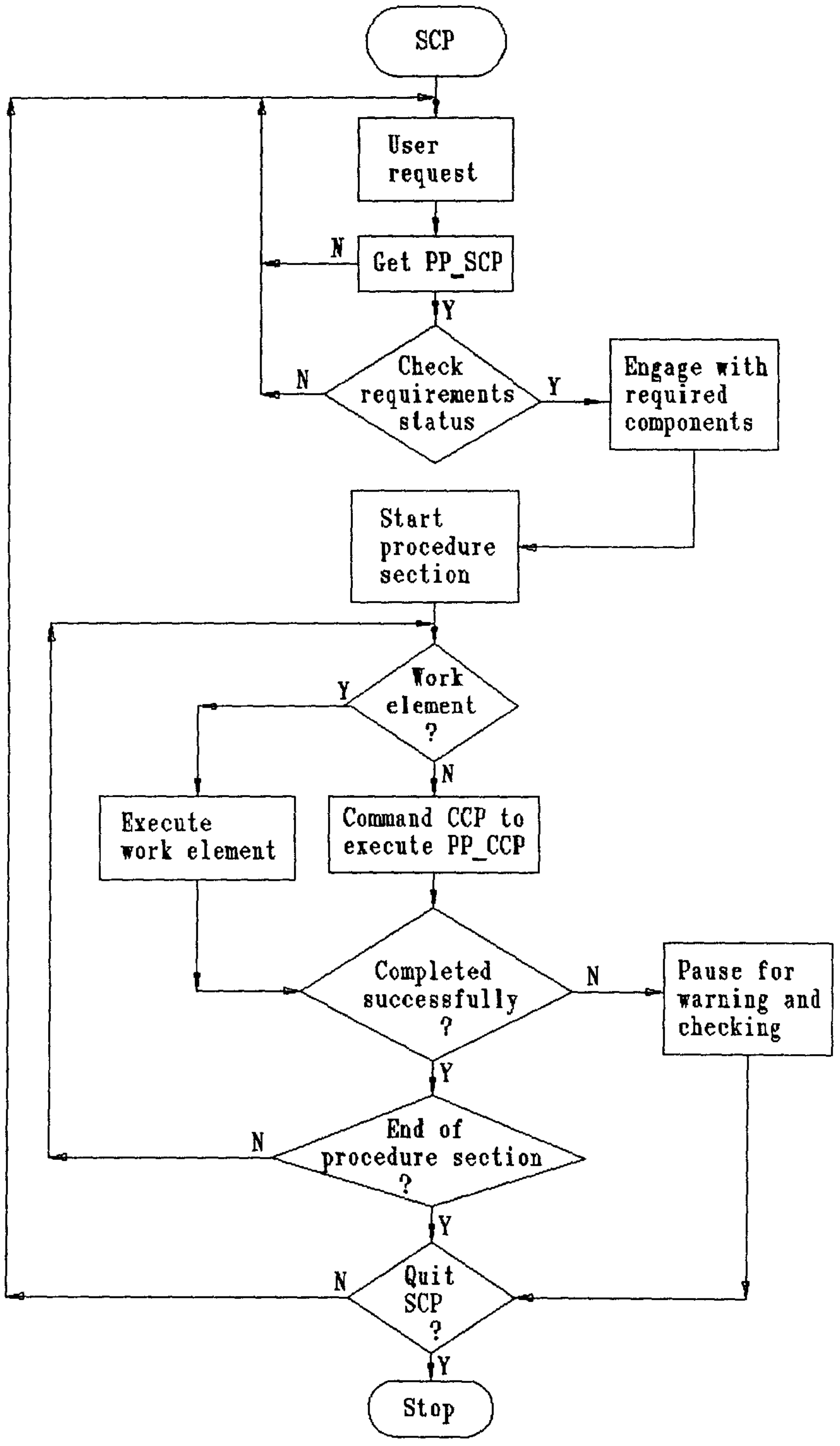

SCP
User request
Get PP_SCP
N
Y
Check requirements status
N
Y
Engage with required components
Start procedure section
Work element ?
Y
N
Execute work element
Command CCP to execute PP_CCP
Completed successfully ?
N
Pause for warning and checking
Y
End of procedure section ?
N
Y
Quit SCP ?
N
Y
Stop

Figure 6.2 System Control Program (SCP)

PP_SCP process plan file. This check will assure that all the FMP/M components required for this manufacturing task are available. It also shows that even if a certain component in the system is down, the system is still operable as long as it is not required by the current manufacturing task. When the SCP enters the procedure section of PP_SCP, it carries out the procedures mentioned in this section one by one. This procedure may be a executable work element or to execute a child process plan. For the latter, it sends the name of the child process plan (PP_CCP) file to the corresponding cell to execute it and wait for the response from that cell. Once this is complete, the SCP then fetches the next procedure and carries it out in a similar fashion until it exhausts the procedure section.

The CCP flowchart is shown in fig. 6.3. In the manual mode, the user can select the target component in order to either command it for services or ask for the status. This is the required function to help the user initializing the cell before entering the auto mode as well as recovering from failure. When in the auto mode, it serves for the SCP to carry out the PP_CCP file as long as the SCP provides the PP_CCP file name. When CCP starts to execute cell level commands in CCFILE, it checks whether the incoming task is to download NC code file into the VMC. The CCP separates this NC code file download function from other cell level

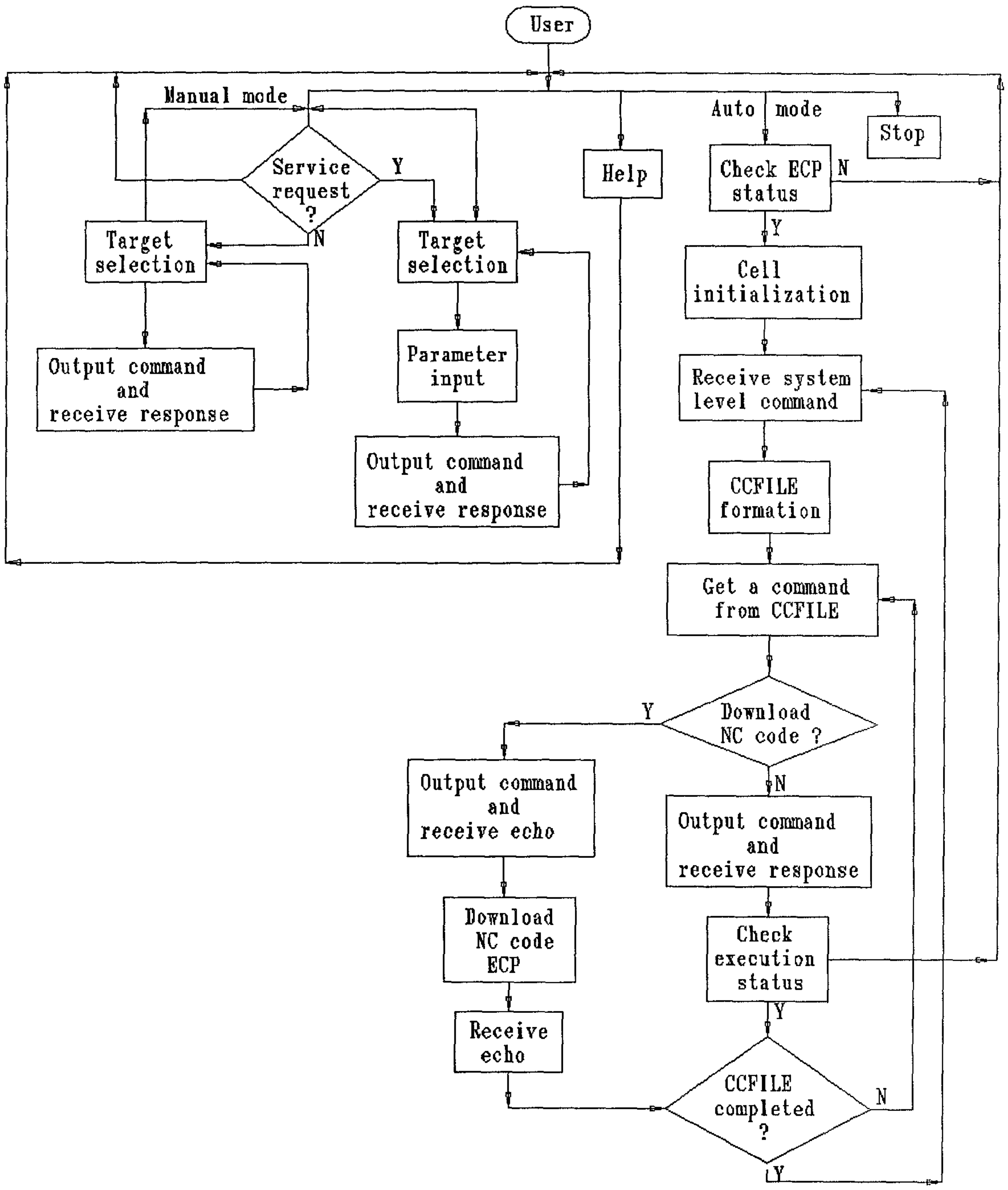


Figure 6.3 Cell Control Program (CCP)

commands in that it utilizes different protocols to communicate with the top-down modules. When executing other cell level commands, the communication protocol is simply send-and-receive type of communication. However, when dealing with NC code downloading, different protocol is used to assure proper file transfer. This protocol will be discussed in the later part of this chapter. Notice that once the system is in the auto mode, it will continue to be in that mode serving for SCP until the system goes wrong and goes back to the root with error messages. Of course, another way to jump out of the auto mode is to reset the CCP.

The ECP flowchart is shown in detail in fig. 6.4. It shows clearly in the auto mode that there are three main branches: COM type of command, STA type of command, and download procedures. This download procedure reveals the downloading protocol in the ECP's part. The ECP uses a memory buffer of size 50K bytes to serve as a virtual disk. The fast fetch/put speed of the virtual disk helps in synchronizing the communication during the NC code downloading procedure.

6.2 Information Flowchart

While starting the development of software for a system with complex functions and structures, a clear idea of how

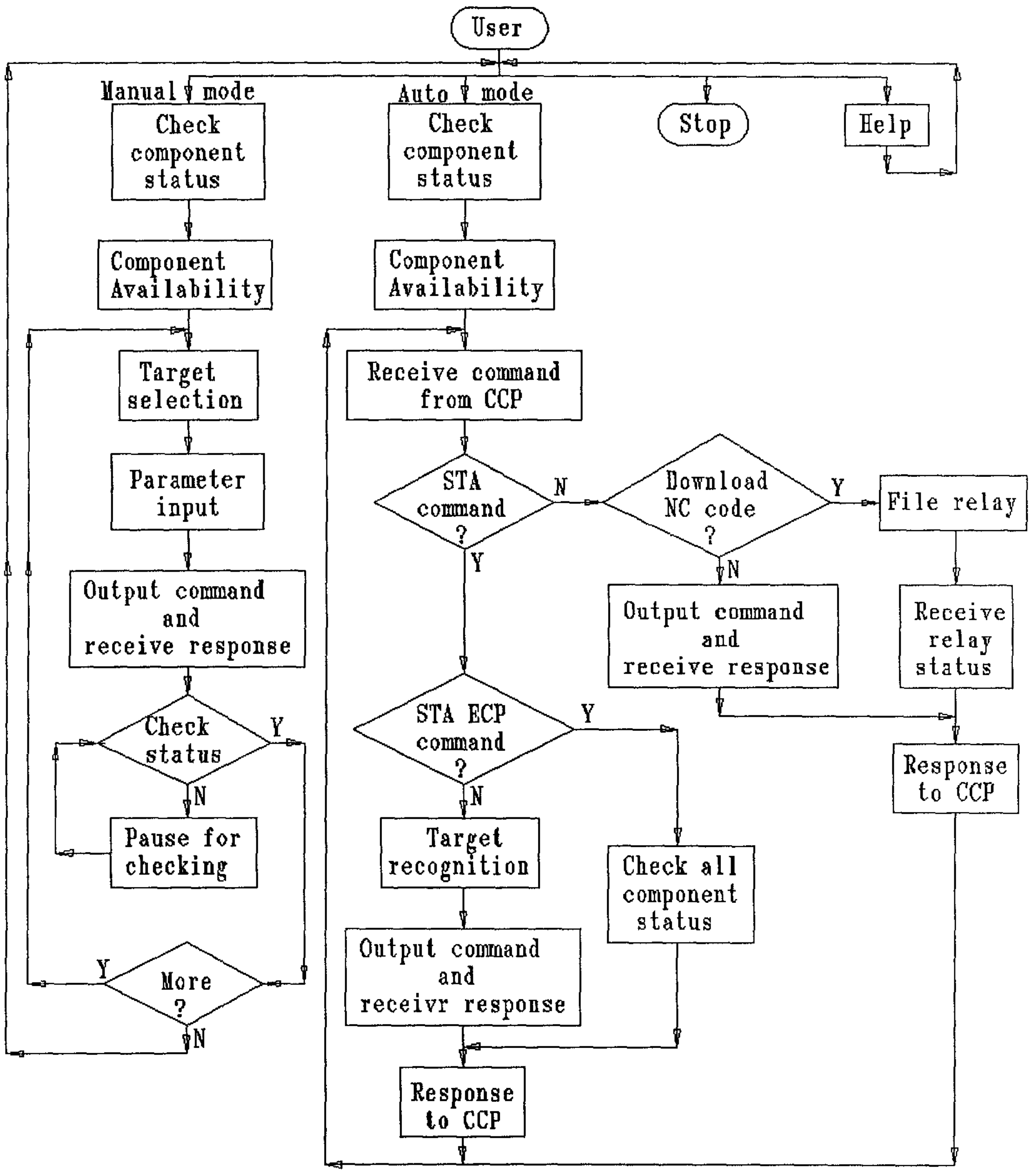


Figure 6.4 Equipment Control Program (ECP)

information flows will be very helpful in keeping the software organized and systematic. Unfortunately, the functional flowchart provides only the sequence of the events occurring during a process but no clue on how the information flows. This is, however, provided by the information flowchart.

Shown in fig. 6.5 and fig. 6.6 are the information flowcharts of CCP and ECP, respectively. These charts show the path of information and also the rules of data processing. In fig. 6.6, for example, the VISCOM (communication subprogram to communicate with vise controller) can only be called by the main program, and it uses BUFR$(2) as the mailbox for incoming and outgoing messages. It also shows that communication subprograms cannot read or write the status buffers. These types of rules are used for structural clarity and for debugging the systems .

6.3 RS-232C Communication Protocol

The RS-232C is a widely used standard of the Electronic Industries Association (EIA). It was originally developed to foster data communication on public telephone networks. The interface to a telephone network is normally made through a device known as a modem. In the 1980s, with the proliferation of microcomputers, terminals are usually

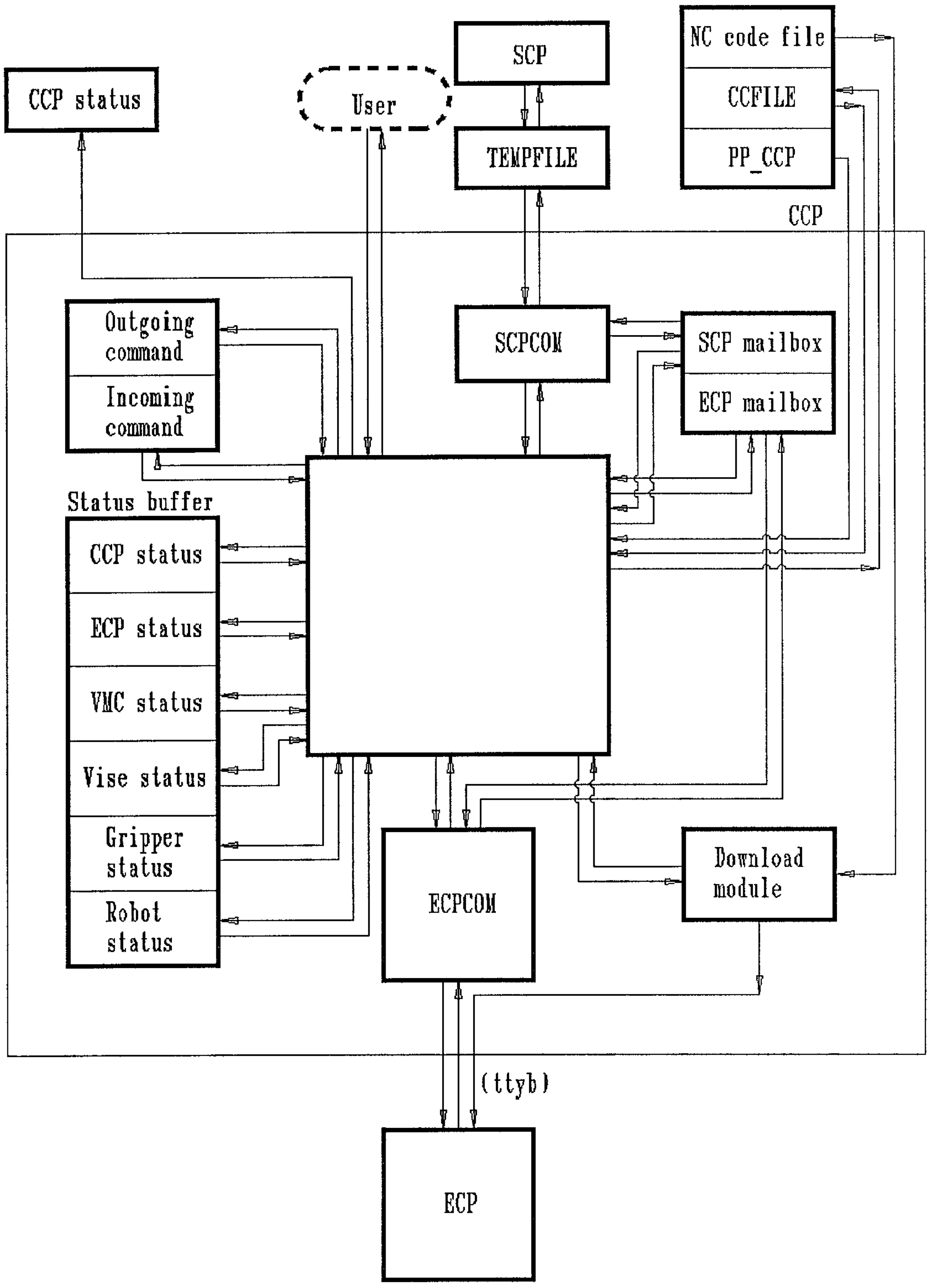


Figure 6.5 CCP information flow

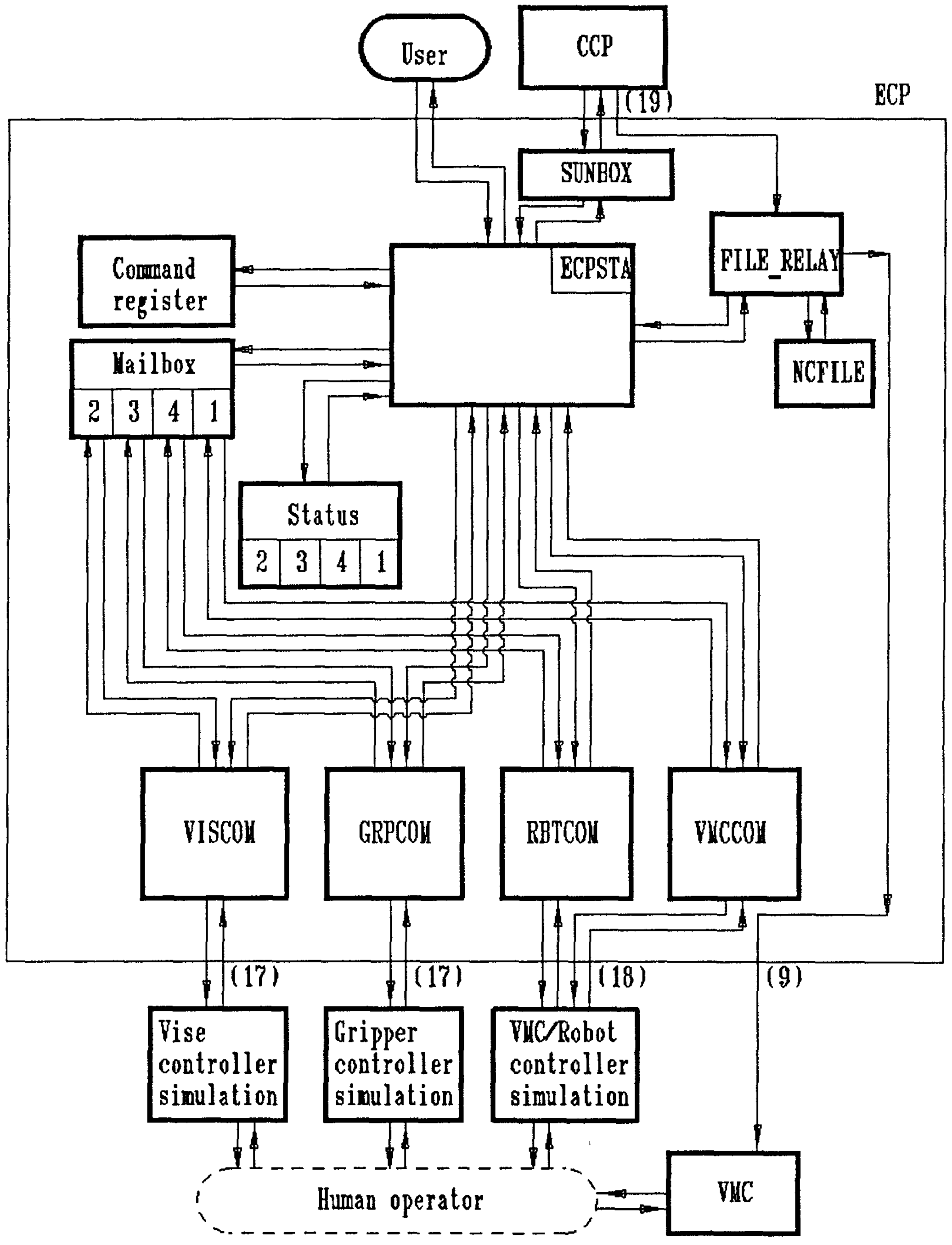


Figure 6.6 ECP information flow

connected directly to computers through RS-232C ports, and do not use modems except for truly remote connections. Since all the computer hardware in FMP is equipped with RS-232C interface, it is the communication interface of choice.

Appendix B shows how the RS-232C connects computers (or terminal) directly or through the use of modem. In the FMP application, however, the receiver is always waiting while the sender sends messages and therefore the hardware flow control of RS-232C is not necessary. To avoid the hardware flow control, the null modem concept is used. Fig. 6.7 shows how this concept is applied in data communication. Null modem connections are used in both ends of each communication link. DTR (Data Terminal Ready) is wired to CD (Carrier Detect) and DSR (Data Set Ready) to provide fake signals to CD and DSR and thus bring the computer to request-and-send procedures. By connecting RTS (Request To Send) and CTS (Clear To Send), the request-and-send procedure is also avoided and the flow of data becomes transparent.

Shown in fig. 6.8 are several types of pin assignments that are used in the FMP. With the null modem pin arrangement, as shown in fig. 6.7 which ignores hardware flow control, we can figure out how different types of pin assignments are linked together to transfer data. Shown in

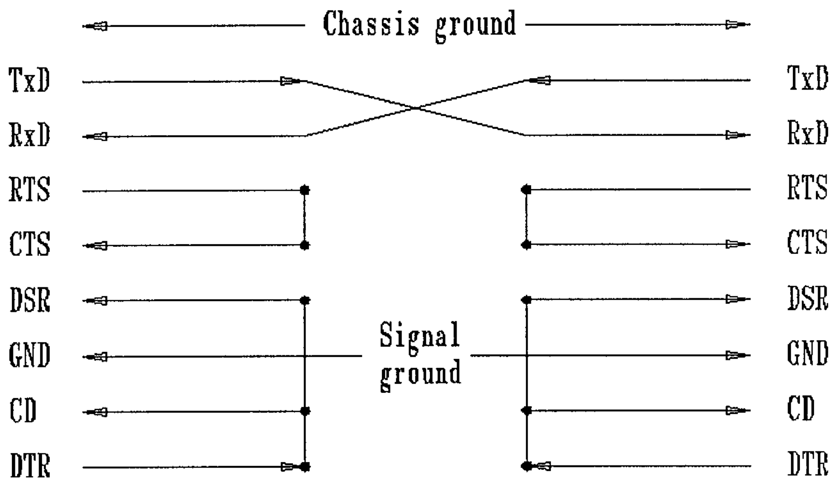
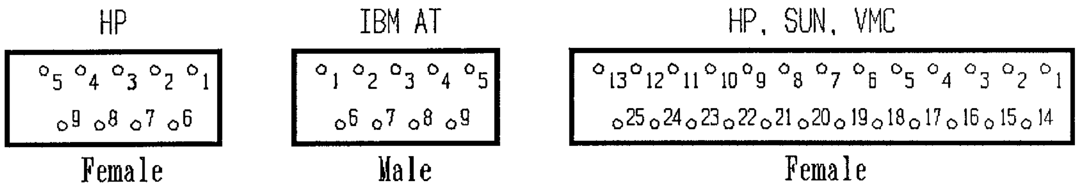


Figure 6.7 Null modem that ignores hardware flow control



(1) DTR	(1) CD	(20) DTR
(2) TxD	(2) RxD	(2) TxD
(3) RxD	(3) TxD	(3) RxD
(4) RTS	(4) DTR	(4) RTS
(5) CTS	(5) GND	(5) CTS
(6) DSR	(6) DSR	(6) DSR
(7) GND	(7) RTS	(7) GND
(8) CD	(8) CTS	(8) CD
(9) RI	(9) RI	(22) RI

Figure 6.8 Types of pin assignments

fig. 6.9 is an example of how the IBM AT is linked with the HP computer.

6.4 NC Code Download Protocol

There are two types of protocols used in this system to transfer messages. One is simply a send-and-receive type as shown in fig. 6.10(a). A and B in each square denote the destination or the origin of the message. In this kind of protocol the receiver is always waiting while the sender sends the message. Once the sender sends out the message, it opens the input port immediately and waits for the incoming message. A typical communication history of completing a cell level command by robot is shown in fig. 6.10(b).

A different type of protocol is used to transfer the NC code files. Since the CCP has the access to NC code file and VMC is the destination, the objects involved in this protocol includes CCP, ECP, and VMC. This download protocol is shown in table 6.1. The protocol was started when the CCP recognized this download command in CCFILE and sent the "DOWNLOAD" message to ECP through channel 19 of HP. The ECP then clears the virtual disk (memory buffer of size 50K) and the previous NC code file in the 3.5"diskette. The ECP then echoes back to the CCP through channel 19 to indicate the "READY" status of ECP.

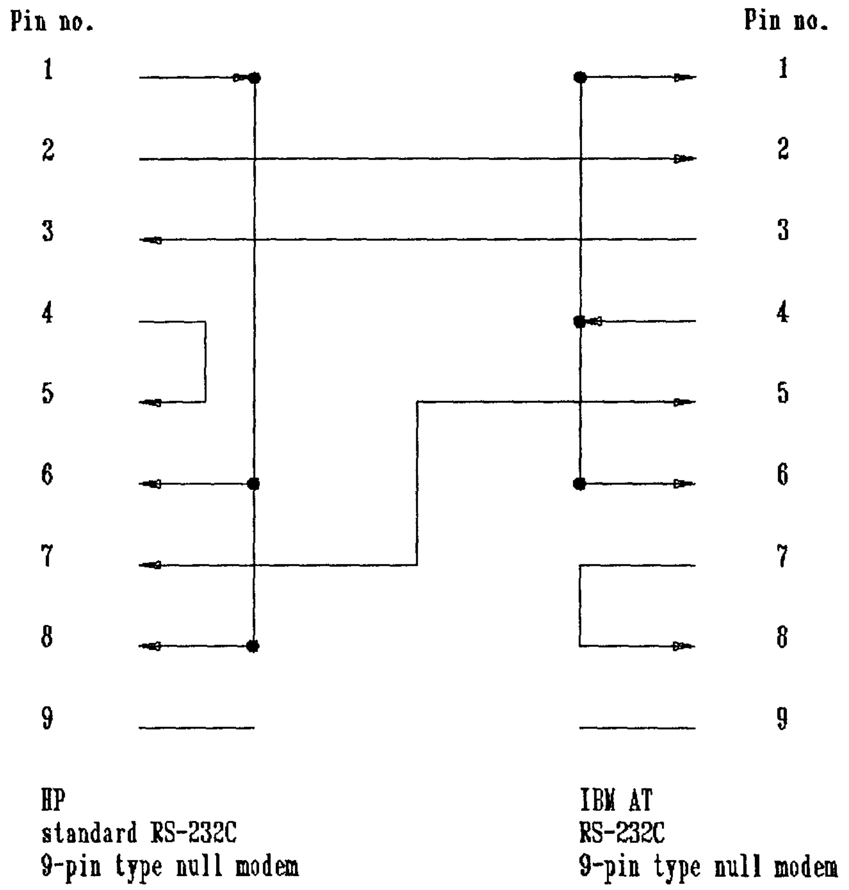
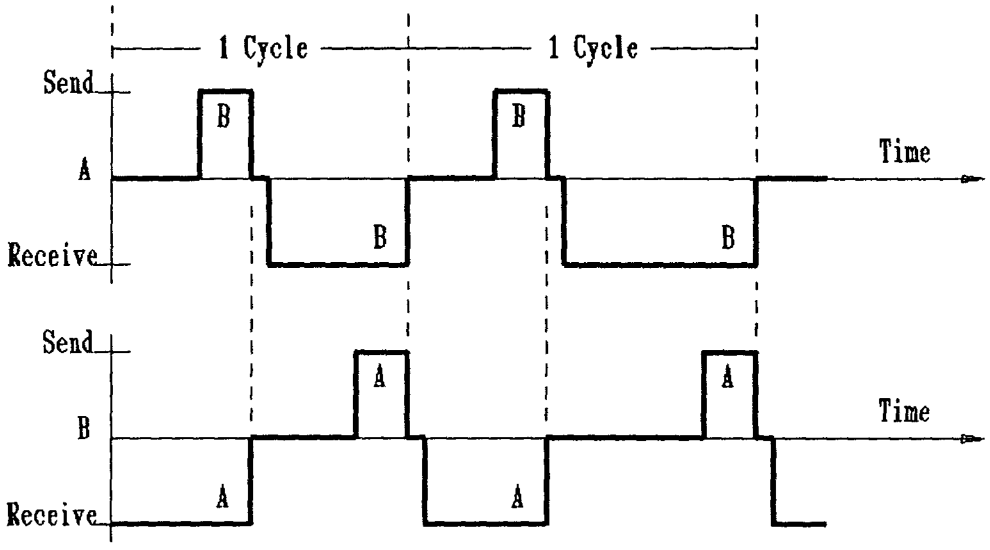
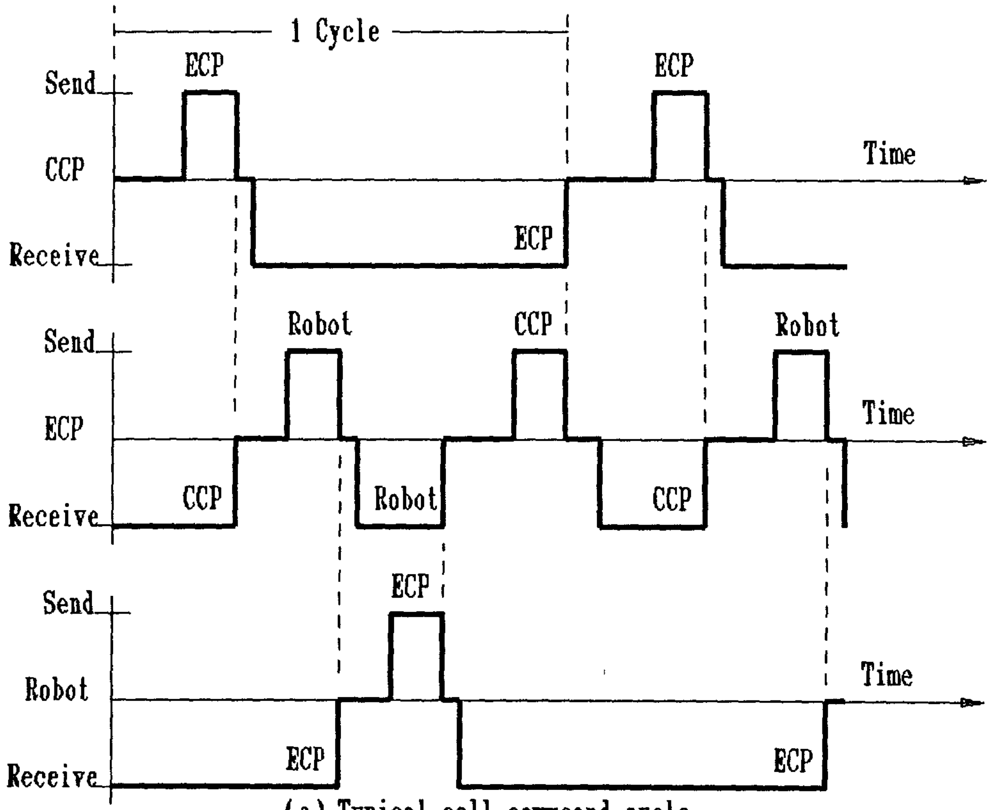


Figure 6.9 HP - IBM AT RS-232C connection



(a) Send-and-receive



(a) Typical cell commcand cycle

Figure 6.10(a), (b) Communication Cycle

STEP	CCP		ECP		VMC
1	DOWNLOAD COMMAND	→			
2		←	DOWNLOAD ECHO		
3	NC CODE	→	(SAVE)		
4			DOWNLOAD COMMAND	→	
5				←	DOWNLOAD ECHO
6			NC CODE	→	
7				←	DOWNLOAD COMPLETE
8		←	DOWNLOAD COMPLETE		

1 DOWNLOAD CYCLE

Table 6.1 Download Procedures

After the CCP receives this message, it starts downloading the NC code file line by line until it reaches the EOF (end of file) of NC code file. The NC code picked up by ECP is saved in the virtual disk and, when the ECP detects the end of receiving, it is dumped into the 3.5" diskette of HP for recording. The ECP then sends the "DOWNLOAD" message to VMC through channel 18 of HP to check whether the VMC is ready to receive the NC code. As soon as the ECP receives the echo from VMC, it starts the downloading process by retrieving the NC code from virtual disk and sending it to VMC through channel 9. At the end of this data transfer, the VMC signals the ECP to indicate the successful transferring. The ECP then sends this message back to CCP and completes the download cycle. The typical communication time history to complete a download cycle is shown in fig. 6.11.

6.5 Integration

Since the FMP system utilizes several kinds of computers and each computer has its own way of packaging a string of characters as well as interpreting them, the integration of the system encounters several communication problems.

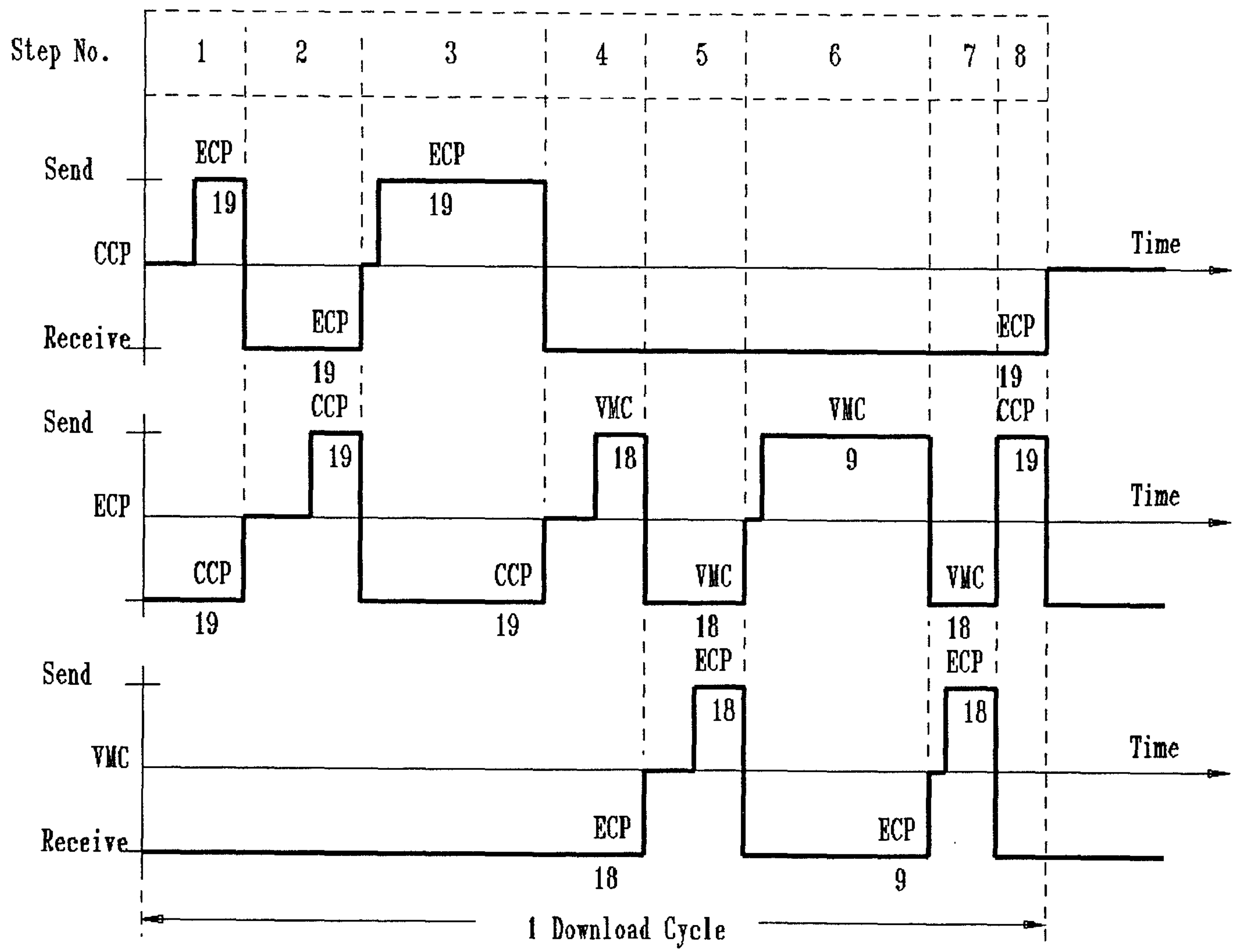
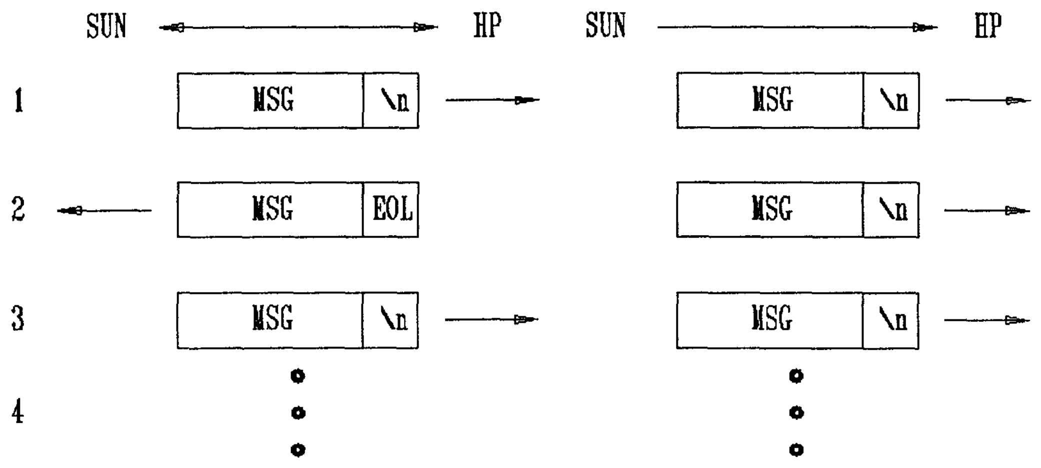


Figure 6.11 Typical time history of a download cycle

Table 6.2 shows one of the main problems. On the left is the bi-directional communication and on the right is the one-way communication. The table shows that the SUN has no problems communicating with HP only during a one-way communication. For bi-directional communication, the HP system dies at the 3rd step. Further investigation reveals that the problem occurs at the beginning of the 3rd step when the ECP (HP) starts enabling the wait-and-receive command "ENTER" for incoming message from SUN.

By tracking the HP I/O status registers 0 to 14 (Appendix C), it is seen that contents of register 6, 10, and 11 are changed at the end of the second step. The content of register 10 indicates that the serial I/O buffer was full. At the beginning of the third step, the HP starts executing the wait-and-receive statement "ENTER" and an error occurs. The reason that causes the error is not clearly understood yet. However, at the end of the second step, the SUN executes a file close statement "close" to close from the reading mode. A possible guess is that a control character is issued before the ttyb (one of the serial I/O ports on the SUN) is actually closed. This issued control character is probably the '\n' character, enters the HP I/O buffer at the beginning of the third step. When the HP executes the "ENTER" command, it reads in the '\n' character into the memory. Unfortunately, it is



STEP \ TYPE	SUN $\longleftrightarrow$ HP	SUN $\longrightarrow$ HP
1	OK	OK
2	OK	OK
3	DIE	OK
4	N/A	OK
	N/A	OK

Table 6.2 Communication error description

characteristic with HP computers to interpret this character as indicating that the sending procedures of SUN have been completed. The buffer pointer of HP then tries to locate and search for characters. Eventually, when the end of the buffer is reached and nothing is found, an error message is given.

This explanation is consistent with the fact that this control character '\n' is not from the first message sent out by SUN (at step 1). It enters the HP buffer shortly after the 2nd step was executed, which is the time that SUN receives the message and is about to close the ttyb from reading mode.

One way to avoid the problem is to use the "RESET" command of HP to clear up all the data in HP's receiver buffer before it executes the 3rd step, the "ENTER" statement. It is also important to make sure that the "RESET" statement is executed by HP after the '\n' character arrives the buffer.

Another major cause of communication problems is the speeds of different computers. Different computers have different speeds and data collisions may occur very easily. To avoid this type of problems, effort has to be put on the investigation of communication time history. The sleep-type statement is used occasionally in software to make the computer sleep in order to achieve asynchronous communication.

CHAPTER 7

CELL IMPLEMENTATION AND VERIFICATION

7.1 Cell Implementation

The software modules and hardware modules of the vertical machining cell have been developed and integrated. However, SCP and UIP are currently not available. In order to link the cell with the FMP/D and make the cell work without changing the communication scheme and information flow, we need a methodology to substitute the SCP to provide the child process plan PP_CCP file name to CCP.

By reviewing the information flow of CCP (fig. 6.5 and fig. 5.2), it is found that the SCPCOM (a communication subprogram in CCP to communicate with SCP) communicates with SCP through the use of a file called "TEMPFILE". In this way, the user can prepare the PP_CCP filename in "TEMPFILE" according to the format mentioned in chapter 5. When the user turns the control of CCP to auto mode, the cell will first check out the cell status and, if the cell is in the "READY" status, it will receive this prepared command from "TEMPFILE" and starts executing the PP_CCP. This also shows the advantage of the software modularity that only the SCPCOM needs to be changed slightly without having to disturb other software modules inside CCP.

7.2 Demonstration

In order to verify the capability of the cell and the FMP automation concept, a prismatic part was designed and then manufactured. Fig. 7.1 shows the demonstration part to be manufactured by the vertical machining cell. This part is designed on the SUN workstation. The list of the PP_CCP process plan file "PP_CCP_TEST" and the NC code file "chendemo_nc" of the demonstration part, are shown in Appendix D. The "VMC_SETUP" file which serves as a library of the vise setup at the machine table is used to help the CCP decide the loading target point of the part in a world coordinates. This is shown in Appendix D as is the CCFILE created by CCP during the manufacturing process.

During the manufacturing process, the computers interact with each other based on the control hierarchy and the cell level commands in the CCFILE. The whole process took about twenty minutes and the part was successfully manufactured. This demonstration shows that the manufacturing information prepared in the FMP/D flows smoothly to the FMP/M and activates the cell to work successfully. It also shows that the process plan files (PP_SCP, PP_CCP) and NC code files carry sufficient information to control and drive the FMP/M components and produce a part.

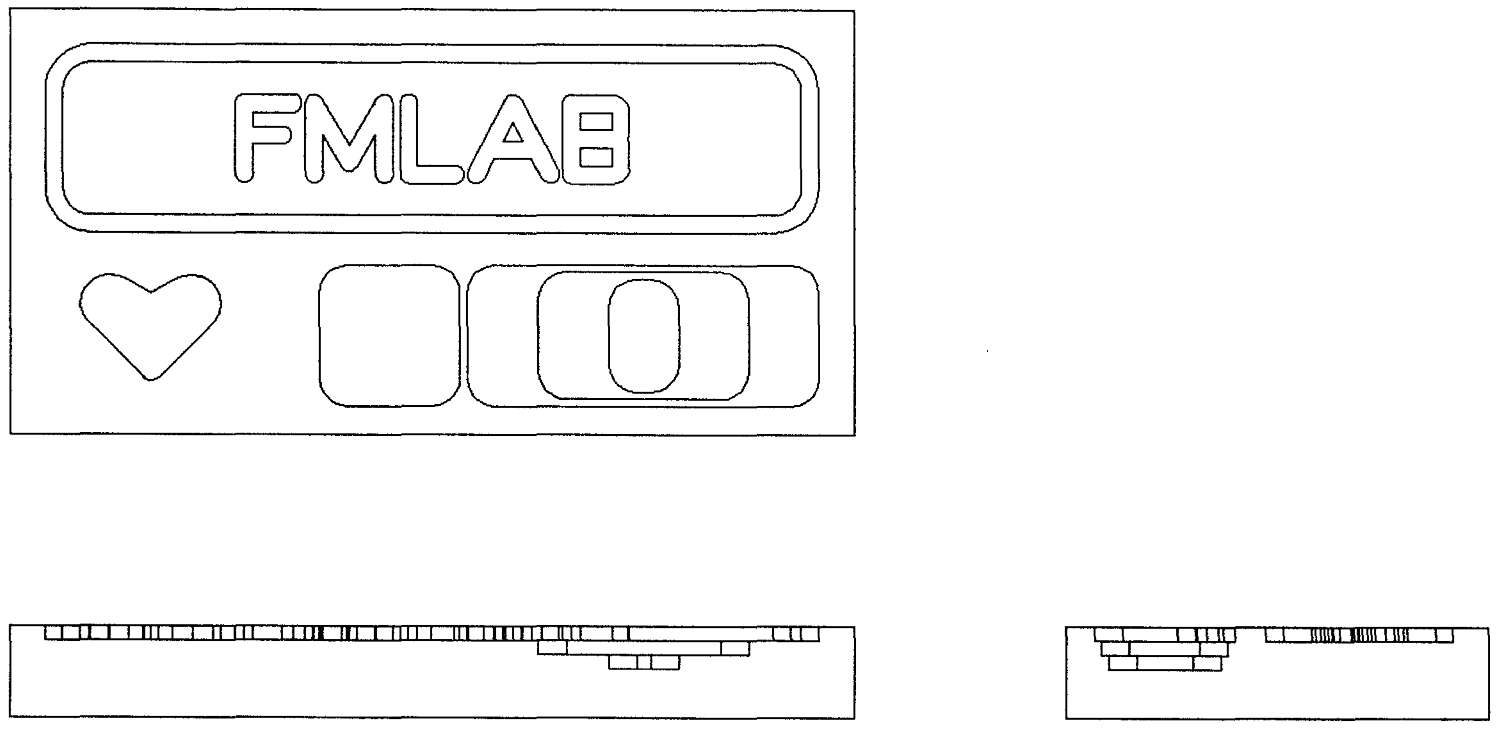


Figure 7.1(a) Demonstration part

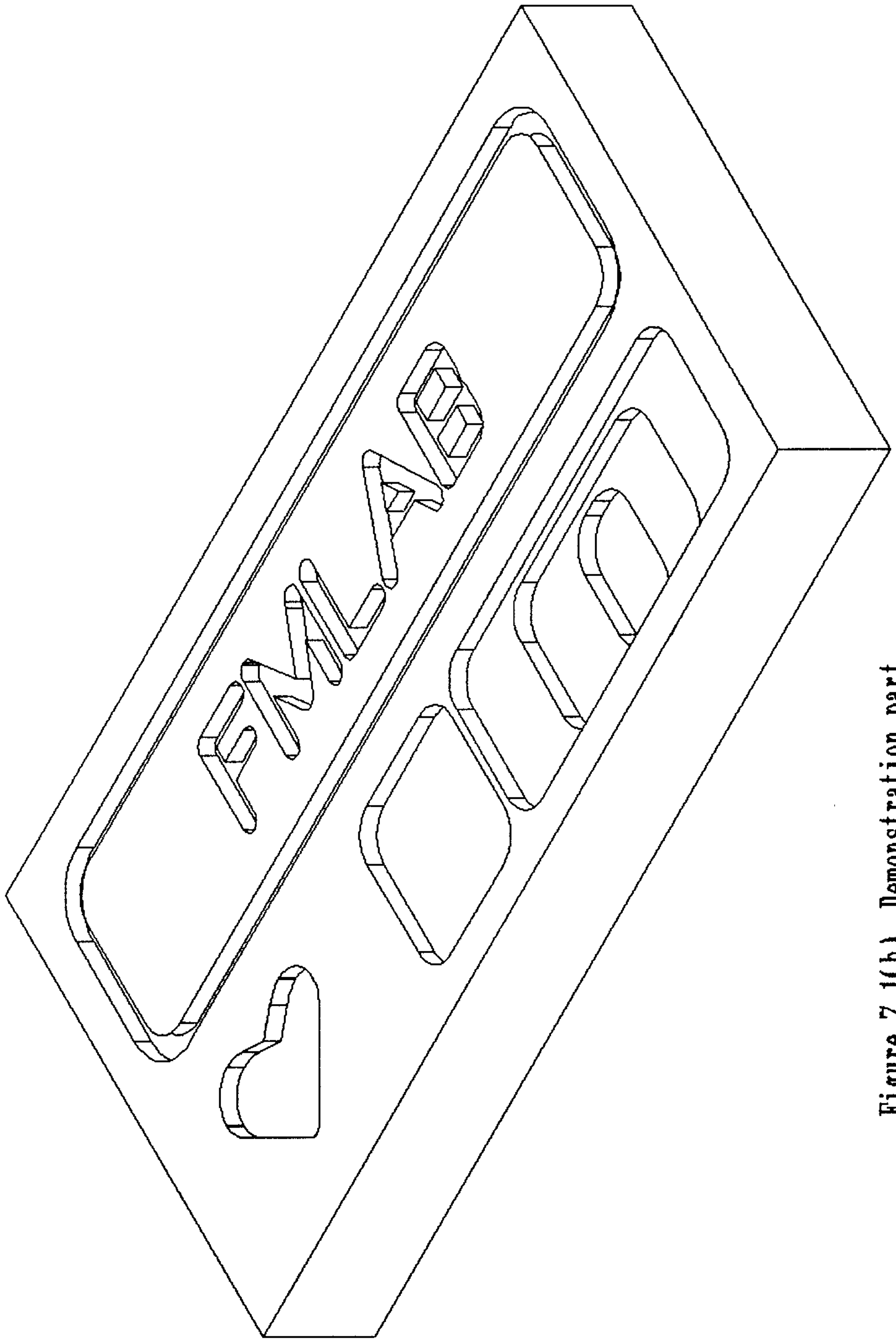


Figure 7.1(b) Demonstration part

CHAPTER 8

CONCLUSIONS AND RECOMMANDATIONS FOR FUTURE WORK

8.1 Conclusions

The Flexible Manufacturing Protocol (FMP) is intended to automate the manufacturing process, which includes design, fixturing considerations, manufacturability checks, process planning, and cell control. It also proposes a way to make efficient use of information resources for computer-integrated production method. Types of files used in the FMP are discussed and identified as drawing file (machine dependent), IGES file, feature file (in Part Model format), intermediate file, process plan file (in Process Plan Format (PPF) of NBS), M & G code file, and NC code file. The use of a User Interface Shell (UIS) is proposed. Software structure for cell control is defined and functions to be achieved by each software module are discussed. A vertical machining cell is built for prismatic part machining using the FMP. The information required by the cell from FMP is identified as process plan files and NC code file.

The successful demonstration of the cell operation verifies the information flow and control scheme inside the cell control module of FMP. The activities between cell components are coordinated properly by the cell host and the part is produced accordingly. The cell is flexible in that it can be linked to another system that provides process plan files and NC code file. Together with the previous work [4, 14, 19, 24, 42, 47], it shows that the development of FMP requires the integration of many finite-state automations which is very important to small-mid size manufacturing systems in helping them move toward production automation in a step-by-step manner.

8.2 Suggestions for Future Work

To fully complete the integration of this cell as well as the FMP, several suggestions are made for future work:

1. Installation and integration of robot and gripper into the cell to replace the robot/gripper controller simulation program.

2. Hardwire the VMC controller and installation of automatic vise to replace the VMC/vise controller simulation program. This can be done by using single computer (controller) to control array actuators.

3. Integration of Data Acquisition System (DAS) into the cell to collect required informations for the above controllers.

4. Development and installation of the feature-based design system for part design.

5. Development of a turning cell for turning part machining.

6. Development of an intelligent process planner, which is capable of doing process plan for turning part and prismatic part, and a System Control Program (SCP) to control the turning cell and vertical machining cell.

7. Integration of an AGV into SCP for material transferring between vertical machining cell, turning cell, and buffer storage.

8. Development and integration of a User Interface Program (UIP) to guide the user in access the FMP.

References

1. Ware Myers, "CAD/CAM: The need for a broader focus", Computer, PP. 105-117, January 1982.

2. Cita M. Furlani, "The Automated Manufacturing Research Facility of National Bureau of Standards", Computer Simulation Conference, July 1983, Vancouver, BC, Canada, (with Ernest W. Kent, Howard M. Bloom and Charles R. McLean).

3. Thomas H. Drake, "Integration of CAD/CAM with Flexible Manufacturing System", 1984, Napier College, Edinburgh, United Kingdom, (With Alex. R. Young).

4. D.K. Anand, "Protocol for Flexible Manufacturing Automation with Heuristics and Intelligence", Manufacturing International, April 1988, Atlanta, Georgia, (With J.A. Kirk, E. Magrab, M. Anjanappa and D. Nau).

5. Joseph C Quinlan, "Computer-Integrated Manufacturing Today, and A Look Ahead", Tooling & Production, PP. 27-36, January 1987.

6. Toshio Sata, "Recent Developments in Computer-Integrated Manufacturing in Japan", Robotics & Computer-Integrated Manufacturing, Vol. 3, No. 4, PP. 373-380, 1987.

7. Y. Ono, "Cell Control Systems", Robotics & Computer-Integrated Manufacturing, Vol. 3, No. 4, PP. 389-393, 1987.

8. Joseph C Quinlan, "Two Robotics Turning Centers riduce WIP by 40 Percent", Tooling & Production, PP. 35-37, May 1987.

9. Robert B. Mills, "CIM Comes of Age", Computer Aided Engineering, PP. 58-62, December 1987.

10. H. -J. Warnecke, "Some Thoughts on Integration in CIM systems", Robotics & Computer-Integrated Manufacturing, Vol. 3, No. 1, PP. 89-95, 1987, (With W. Dangelmaier).

11. M. Weck, "Applicability of Expert Systems to Flexible Manufacturing", Robotics & Computer-Integrated Manufacturing, Vol. 3, No. 1, PP. 97-103, 1987, (With G.Kiratli).

12. H. Hammer, "Flexible Manufacturing Cells and Systems with Computer Intelligience", Robotics and Computer-

Integrated Manufacturing, Vol. 3, No. 1, PP. 39-54, 1987.

13. D. Reisch, "Total CAM Concept Embracing Logistics", Robotics & Computer-Integrated Manufacturing, Vol. 3, No. 1, PP. 105-122, 1987.

14. Rajiv Uppal, "Flexible Manufacturing Protocol Driver for a Vertical CNC Machining Center", M.S. Thesis, Mechanical Engineering, College Park, University of Maryland, December 1986.

15. W. Gutschke, "CIM: Competitive Edge in Manufacturing", Robotics & Computer-Integrated Manufacturing, Vol. 3, No. 1, PP. 77-87, 1987, (With K. Mertins).

16. Steven R. Ray, "A Knowledge Representation Scheme for Processes in an Automated Manufacturing Environment", Center for Manufacturing Engineering, 1987, National Bureau of Standards, Gaithersburg, MD.

17. D. K. Anand, "Research in the Flexible Manufacturing Laboratory", The Systems Research Center, SRC-TR-86-61, 1986, College Park, University of Maryland.

18. James S. Albus, "Programming a Hierarchical Robot Control System", 12th International Symposium on

Industrial Robots, June 1982, Paris, France, (With Anthony J. Harbera and M.L. Fitzgerald).

19. B. Kumar, "Integration and Testing of a Intelligent Feature Extractor within a Flexible Manufacturing Protocol", Proceedings of the 16th NAMRC, May 1988, Urbanan, IL, (With D.K. Anand and J.A. Kirk).

20. Vitan, "A Modular Approach to Flexible Automation", Manufacturing Engineering, M. Sc., ITCR, Bulgaria

21. Toshio Sata, "Functions Required for Advanced Flexible Manufacturing Systems", Robotics & Computer-Integrated Manufacturing, Vol. 3, No. 4, PP. 417-421, 1987, (With H. Hiraoka and M. Miki).

22. Pual C Miller, "Automatic Setup for Machining Centers", Tooling & Productions, PP. 68-70, May 1987.

23. Robert K. Southard, "LANs - Nervous System for Your Factory", Tooling & Production, PP. 26-32, December 1987

24. D.K. Anand, "Super Computer and Hierarchical Control: A System Point of View", Proceeedings, NSF Conference on Supercomputers in Mechanical Systems Research, 1984, Lawrenec, Livermore National Laboratory, CA., (With J.A. Kirk, M. Anjanappa and M. Pecht).

25. Mark R. Cutkosky,"Precision Flexible Manufacturing Cells within a Manufacturing System", Technical Report, Carnegie-Mellon University, 1983, Pittsburgh, PA., (With Paul S. Fussell and Robert Milligan, Jr.).

26. G. Spur, "Cell Concepts for Flexible Automated Manufacturing", Journal of Manufacturing Systems, Vol. 5, No. 3, PP. 171-179, 1987, (With G. Seliger and B. Viehweger).

27. Albert T. Jones, "A Proposed Hierarchical Control Model for Automated Manufacturing System", Journal of Manufacturing Systems, Vol. 5, No. 1, PP. 15-25, 1986, (With Charles R. McLean).

28. Paul Prickett, "An Approach to CADCAM implementation", Computer-Integrated Manufacturing Journal, PP. 245-249, December 1987.

29. Don Ralston, "Computer Integrated Manufacturing", Computer-Aided Engineering Journal, PP. 167-175, August 1987, (With Tony Munton).

30. Edward B. Magrab, "Vertical Machining Workstation of the AMRF: Equipment Integration", Center for Manufacturing

Engineering, National Bureau of Standards, 1986, Gaithersburg, MD.

31. Charles R. McLean, "The Vertical Workstation of the AMRF: Software Integration", ASME Symposium on Intelligent and Integrated Manufacturing, September, 1986, Chicago, Illinois.

32. Kaare Christian, "The Unix Operating System", ISBN 0-471-87542-2, John Wiley & Sons, Inc., 1983.

33. Marc J. Rochkind, "Advanced Unix Programming", ISBN 0-13-0118184, Prentice-Hall, Inc., 1985.

34. V. Carl Hamacher, "Computer Organization", ISBN 0-07-025683-7, McGraw-Hill, Inc., 1984, (With Zvonko G. Vranesic and Sofwat G. Zaky).

35. Harold S. Stone, "Microcomputer Interfacing", ISBN 0-201-07403-6, Addison-Wesley Publishing, Inc., 1982.

36. M. Morris Mano, "Computer System Architecture", 2nd Edition, ISBN 0-13-166611-8, Prentice-Hall, Inc., 1982.

37. Mikell P. Groover, "CAD/CAM Computer-Aided Design and Manufacturing", ISBN 0-13-110130-7, Prentice-Hall, Inc., 1984, (With Emory W. Zimmers, Jr.).

38. Edward J. Preston, "CAD/CAM Systems", ISBN 0-8247-7257-1, Marcel Dekker, Inc., 1984, (With George W. Grawford and Mark E. Coticchia).

39. Al Kelley, "A Book on C", ISBN 0-8053-6860-4, Benjamin/Cummings Publishing, Inc., 1984, (With Ira Phol).

40. Brian W. Kernighan, "The C Programming Language", ISBN 0-13-110163-3, Prentice-Hall, Inc., 1978, (With Dennis M. Ritchie).

41. E. Paul DeGarmo, "Materials and Processes in Manufacturing", 6th Edition, ISBN 0-02-328620-2, Macmillan, Inc., 1984, (With J. Temple Black and Ronald A. Kohsor).

42. J.A. Kirk, "Implementation of a Flexible Manufacturing Protocol", Proceedings of the IASTED Applied Control and Identification Conference, December 1986, Los Angeles, CA., (With D.K. Anand, M. Anjanappa, and R. Uppal).

43. "Long Range Plan for the IGES Organization", Release 1, National Bureau of Standards, March 1986, Gaithersburg, MD.

44. Bradford Smith, "Initial Graphics Exchange Specification - The IGES Project - A Status Report", National Bureau of Standards, December 1985, Gaithersburg, MD.

45. Bradford M. Smith, "Product Data Exchange Specification - The PDES Project - Objectives, Plans and Schedules", National Bureau of Standards, June 1986, Gaithersburg, MD.

46. Walter Kellner Rickert, Jr., "The Use of IGES in Automated CNC Machining", M.S. Thesis, Mechanical Engineering, College Park, University of Maryland, Feburary 1987.

47. B. Kumar, "An Intelligent Feature Extractor for Automated Machining", Proceedings of the 5th International Conference on System Engineering, September 1987, Dayton, Ohio, (With D.K. Anand and J.A. Kirk).

APPENDIX A

A BRIEF OVERVIEW OF THE NBS PROCESS PLAN FILE FORMAT

APPENDIX A

A BRIEF OVERVIEW OF THE NBS PROCESS PLAN FILE FORMAT

The NBS Process Plan File (PPF) format is described in Backus-Naur (BNF) notation. To understand the PPF format, it is necessary to first introduce the BNF notation. A brief description of the BNF notation is shown in the following :

1. <xx>   Denotes a non-terminal symbol whose name is xx. A "non-terminal" is a symbol of the BNF notation which can be further decomposed into a set of non-terminals and/or terminals. Eventually, all symbols are decomposed into terminals. A "terminal" is a symbol or character of the object language. The "object language" consists of all symbols and characters that will appear in the actual file.

2. ::=   The non-terminal to the left of this symbol is composed of all those elements that are to the right of this symbol (expresses decomposition).

3. <xx> <yy>   This expresses concatenation of two non-terminals ("and"). The concatenation applies to whatever these non-terminals decompose to as well.

4. | This means "or" (any one of the specified elements may be chosen to place in this position).

5. { }   Means zero or more occurrences of. For example, {<header_line>} means the same as <header_line> ... <header_line>.

6. [ ]   Means optional.

7. xx .. yy   This notation is for numerical or alphabetic ranges.

8. <???> Means as yet undefined.

A simple example of BNF is given:

```
<a> ::= <b> <c> !
<b> ::= hi
<c> ::= there
```

means <a> is equal to "hi there !".

Comments:

1. Keywords, values, and parameters are to be 19 characters or less.

2. All letters are uppercase.

3. Any terminal (punctuation_mark, integer) may be preceded and followed by whitespace unless otherwise specified.

4. The notation xxH in the following BNF represents the hexadecimal number specifying an ascii character. For instance, 2H means space (chr$(32)).

5. Any element may be omitted by delineating with the proper punctuation. For instance, in order to specify no precedence steps, two consecutive semi-colons may be used.

THE PROCESS PLAN SPECIFICATION IN BNF

```
<pp_file> ::= --PROCESS_PLAN--
                    <header_section>
                    <parameters_section>
                    <rqmts_list>
                    <procedure_specification>
              --END_PROCESS_PLAN--

<header_section> ::= --HEADER_SECTION--
                          {<header_line>}
                     --END_HEADER_SECTION--

     <header_line> ::= <header_elem_name> = <value>
     <header_elem_name> ::= <keyword>

<parameters_section> ::= --PARAMETERS_SECTION--
                              {<parm_line>}
                         --END_PARAMETERS_SECTION--

     <parm_line> ::= <parm_name> ; <parm_type> ;
                     <parm_range> ; <parm_default>
     <parm_name> ::= $$<keyword>
     <parm_type> ::= <???>
     <parm_range> ::= <value> , <value>
     <parm_default> ::= <???>
```

```
<rqmts_list> ::= --REQUIREMENTS_SECTION--
                         {<rqmt_line>}
                 --END_REQUIREMENTS_SECTION--


     <rqmt_line> ::= <rqmt_number> ; <rqmt_identifier> ;
                     <rqmt_type> ; <rqmt_description> ;
                     <rqmt_quantity> ; <parent_rqmts>
     <rqmt_number> ::= <integer>
     <rqmt_identifier> ::= <keyword>
     <rqmt_type> ::= <keyword>
     <rqmt_description> ::= <???>
     <rqmt_quantity> ::= <integer>
     <parent_rqmts> ::= <rqmt_line_num_list>
     <rqmt_line_num_list> ::= {<keyword> ,} <keyword>


<procedure_specification> ::= --PROCEDURE_SECTION--
                                   {<procs_line>}
                              --END_PROCEDURE_SECTION--


     <procs_line> ::= <step_number> ; <work_descr> ;
                      <prec_steps> ; <duration>
     <step_number> ::= <integer>
     <work_descr> ::= <work_element_name>
                      {, <keyword> = <value>}
     <work_element_name> ::= <keyword>
     <prec_steps> ::= {<integer> ,} <integer>
     <duration> ::= " <days> : <hrs> : <min> : <sec> "
```

```
          (No whitespace allowed between characters)
     <days> ::= <digit> <digit> <digit> <digit>
          (No whitespace allowed between digits)
     <hrs> ::= 00 .. 23
     <min> ::= 00 .. 59
     <sec> ::= 00 .. 59

<keyword> ::= <keyword_prefix> {<uppercase_letter> | <digit>
              | $ | # | _ | - | % | & | + | ! | @ | *}
          (No whitespace allowed between characters)
<keyword_prefix> ::= <uppercase_letter> | $ | # | @
<value> ::= <number> | <keyword> | <string>
<string> ::= " {<ascii_printable_char>} "
<whitespace> ::= { CR | LF | SPACE | TAB | FORMFEED }
<upper_case_letter> ::= A .. Z (41H .. 5AH)
<digit> ::= 0 ..9 (30H .. 39H)
<integer> ::= <digit> | <digit> <digit>
          (No whitespace allowed between digits>
<ascii_printable_char> ::= SPACE .. ~ | TAB | FORMFEED | CR
                           | VT | LINEFEED (20H .. 7EH | 12H
                           | 10H | 9H | 0BH | 0AH)
          (" or 22H must be preceded by \, or 5CH and also
          \, or 5CH)
<file_keyword> ::= --PROCESS_PLAN-- | --END_PROCESS_PLAN--
<section_keyword> ::= --PARAMETERS_SECTION-- |
                      --END_PARAMETERS_SECTION-- |
                      --HEADER_SECTION-- |
                      --END_HEADER_SECTION-- |
                      --REQUIREMENTS_SECTION-- |
                      --END_REQUIREMENTS_SECTION-- |
                      --PROCEDURE_SECTION-- |
                      --END_PROCEDURE_SECTION--
<punctuation_mark> ::= ; | = | . | : | ,
          (3BH | 3DH | 2EH | 3AH | 2CH)
<number> ::= <integer> | <integer>.<integer> |
             <integer>[.<integer>]E<exponent>
          (No whitespace allowed between characters)
<exponent> ::= [+]<integer> | -<integer>
          (No whitespace allowed between characters)
```

Example Process Plan:

```
--PROCESS_PLAN--

--HEADER_SECTION--
     PLAN_ID                := PP-CELL-1;
     PLAN_VERSION           := 1;
     PLAN_TYPE              := ROUTING-SLIP;
     PLAN_NAME              := "FILTER-HOUSING";
     PROCESS-ENGINEER       := "Peter"
     PART-NUMBER            := 31;
     ENG-DRAW-#             :=123987;
--END_HEADER_SECTION--

--PARAMETERS_SECTION--
     $$TRAY001              : PART-TRAY
     $$TRAY002              : TOOL-TRAY
     $$LOT001               : LOT;
     $$TOOL-SET001          : TOOL-SET;
--END_PARAMETERS_SECTION--

--REQUIREMENTS_SECTION--
<<1>> PROCESS-PLAN
          ( PLAN-ID      => PP-MHS-1          ,
            PLAN-VERSION => 1                 ,
            PLAN-TYPE    => OPERATION-SHEET   ,
            PLAN-NAME    => "FILTER-HOUSING" );
```

```
<<2>> PROCESS-PLAN
          ( PLAN-ID      => PP-MHS-2          ,
            PLAN-VERSION => 1                 ,
            PLAN-TYPE    => OPERATION_SHEET   ,
            PLAN-NAME    => "FILTER-HOUSING" );
<<3>> PROCESS-PLAN
          ( PLAN-ID      => PP-MHS-3          ,
            PLAN-VERSION => 1                 ,
            PLAN-TYPE    => OPERATION-SHEET   ,
            PLAN-NAME    => "FILTER-HOUSING" );
<<4>> PROCESS-PLAN
          ( PLAN-ID      => PP-MHS-4          ,
            PLAN-VERSION => 1                 ,
            PLAN-TYPE    => OPERATION-SHEET   ,
            PLAN-NAME    => "FILTER-HOUSING" );
<<5>> PROCESS-PLAN
          ( PLAN-ID      => PP-VWS-1          ,
            PLAN-VERSION => 1                 ,
            PLAN-TYPE    => OPERATION-SHEET   ,
            PLAN-NAME    => "FILTER-HOUSING" );
<<6>> PROCESS-PLAN
          ( PLAN-ID      => PP-VWS-2          ,
            PLAN-VERSION => 1                 ,
            PLAN-TYPE    => OPERATION-SHEET   ,
            PLAN-NAME    => "FILTER-HOUSING" );
<<7>> PROCESS-PLAN
          ( PLAN-ID      => PP-VWS-3          ,
            PLAN-VERSION => 1                 ,
            PLAN-TYPE    => OPERATION-SHEET   ,
            PLAN-NAME    => "FILTER-HOUSING" );
<<8>> WORKSTATION
          ( WORKSTATION-ID    => VWS    );
<<9>> WORKSTATION
          ( WORKSTATION-ID    => MHS    );
<<10>> TRAY
          ( TRAY-TYPE    => SECTOR-4    ,
            TRAY-ID      => $$TRAY001   );
<<11>> TRAY
          ( TRAY-TYPE    => SECTOR-4    ,
            TRAY-ID      => $$TRAY002   );
--END_REQUIREMENTS_SECTION--


--PROCEDURE_SECTION--
<<1>> DELIVER-TRAY
          ( ORIGIN       => MHS         ,
            DESTINATION  => VHS         ,
            TRAY-TYPE    => SECTOR-4    ,
            TRAY-ID      => $$TRAY001   ,
            SYSTEM       => MHS         ,
            TYPE         => COMPLEX     ,
            PLAN-ID      => PP-MHS-1    ,
            PREC-STEPS   => ()          ,
            TIME         => 0000:00:00:30 );
<<2>> DELIVER-TRAY
```

```
          ( ORIGIN        => MHS              ,
            DESTINATION   => VHS              ,
            TRAY-TYPE     => SECTOR-4         ,
            TRAY-ID       => $$TRAY002        ,
            SYSTEM        => MHS              ,
            TYPE          => COMPLEX          ,
            PLAN-ID       => PP-MHS-2         ,
            PREC-STEPS    => ()               ,
            TIME          => 0000:00:00:30 );
<<3>> RECEIVE-TRAY
          ( TRAY-TYPE     => SECTOR-4         ,
            TRAY-ID       => $$TRAY001        ,
            SYSTEM        => VWS              ,
            TYPE          => PRIMITIVE        ,
            PREC-STEPS    => (1)              ,
            TIME          => 0000:00:00:30 );
<<4>> RECEIVE-TRAY
          ( TRAY-TYPE     => SECTOR-4         ,
            TRAY-ID       => $$TRAY002        ,
            SYSTEM        => VWS              ,
            TYPE          => PRIMITIVE        ,
            PREC-STEPS    => (2)              ,
            TIME          => 0000:00:00:30 );
<<5>> SETUP-AREA
          ( AREA-ID       => TOOL-CHANGER          ,
            ITEMS         => $$TOOL-CHANGER        ,
            SYSTEM        => VWS                   ,
            TYPE         => COMPLEX          ,
            PLAN-ID      => PP-VWS-1         ,
            PREC-STEPS   => (4)              ,
            TIME         => 0000:00:02:45    );
<<6>> MACHINE-LOT
          ( LOT-ID       => $$LOT001         ,
            LOT-TYPE     => FILTER-HOUSING   ,
            QUANTITY     => 4                ,
            TRAY-ID      => $$TRAY001        ,
            TOOL-SET     => $$TOOL-SET001    ,
            SYSTEM       => VWS              ,
            TYPE         => COMPLEX          ,
            PLAN-ID      => PP_VWS-2         ,
            PREC-STEPS   => (3,5),           ,
            TIME         => 0000:00:28:15    );
<<7>> TAKEDOWN-AREA
          ( AREA-ID      => TOOL-CHANGER     ,
            ITEMS        => $$TOOL-SET-1     ,
            SYSTEM       => VWS              ,
            TYPE         => COMPLEX          ,
            PLAN-ID      => PP-VWS-3         ,
            PREC-STEPS   => (6)              ,
            TIME         => 0000:00:02:30    );
<<8>> SHIP-TRAY
          ( TRAY-TYPE    => SECTOR-4     ,
            TRAY-ID      => $$TRAY001    ,
            SYSTEM       => VWS          ,
              TYPE          => PRIMITIVE     ,
              PREC-STEPS    => (6)           ,
              TIME          => 0000:00:00:30 );
<<9>> SHIP-TRAY
            ( TRAY-TYPE     => SECTOR-4      ,
              TRAY-ID       => $$TRAY002     ,
              SYSTEM        => VWS           ,
              TYPE          => PRIMITIVE     ,
              PREC-STEPS    => (7)           ,
              TIME          => 0000:00:00:30 );
<<10>> DELIVER-TRAY
            ( ORIGIN        => VWS           ,
              DESTINATION   => MHS           ,
              TRAY-TYPE     => SECTOR-4      ,
              TRAY-ID       => $$TRAY001     ,
              SYSTEM        => MHS           ,
              TYPE          => COMPLEX       ,
              PLAN-ID       => PP-MHS-3      ,
              PREC-STEPS    => (8)           ,
              TIME          => 0000:00:00:30 );
<<11>> DELIVER-TRAY
            ( ORIGIN        => VWS           ,
              DESTINATION   => MHS           ,
              TRAY-TYPE     => SECTOR-4      ,
              TRAY-ID       => $$TRAY002     ,
              SYSTEM        => MHS           ,
              TYPE          => COMPLEX       ,
            PLAN-ID       => PP-MHS-4      ,
            PREC-STEPS    => (9)           ,
            TIME          => 0000:00:00:30 );
--END_PROCEDURE_SECTION--


--END_PROCESS_PLAN--
```

APPENDIX B

THE RS-232C INTERFACE FOR ASYNCHRONOUS COMMUNICATION

APPENDIX B

THE RS-232C INTERFACE FOR ASYNCHRONOUS COMMUNICATION

The EIA RS-232C completely specifies the interface between data communication devices (for example, modems) and data terminal equipment (for example, the computer and the I/O terminal). The standard RS-232C interface consists of 25 connection points as shown in table B.1.

Fig. B.1 shows how the RS-232C interface two communication equipments. Conspicuous in the figure are the two grounds defined in the RS-232C standard. One ground is a chassis ground that is tied directly to the shields in the systems and is also called shield ground. This ground connection should be made between two devices only if it is safe to connect the chassis grounds together. The other ground is a signal ground that provides a common reference point for all other signals. This connection is mandatory. Because the signal grounds are not necessarily isolated from the chassis ground, RS-232C has an inherent potential ground-loop problem. While the standard is quite useful for short distances, for longer distances it becomes unreliable and hazardous. The published standard recommends that each device should have a cable not in excess of 50 feet. The terminal/computer interface on the left and the modem on the right have a pair of wires dedicated to Transmitted Data

TABLE B.1 THE RS-232C CONNECTOR STANDARD

Pin #	EIA	Function
1	AA	Protective ground (shield)
2	BA	Transmitted data
3	BB	Received data
4	CA	Request to send
5	CB	Clear to send
6	CC	Data set ready
7	AB	Signal ground (common return)
8	CF	Received line signal detector (carrier detected)
9	--	Reserved for testing
10	--	Reserved for testing
11	--	Unassigned
12	SCF	Secondary received line signal detector
13	SCB	Secondary clear to send
14	SBA	Secondary transmitted data
15	DB	Transmitter signal element timing (terminal transmitter clock)
16	SBB	Secondary received data
17	DD	Receiver signal element timing (modem receiver clock)
18	--	Unassigned
19	SCA	Secondary request to send
20	CD	Data terminal ready
21	CG	Signal quality detector
23	CH	Data signal rate selector (DTE to DCE)
	CI	Data signal rate selector (DCE to DTE)
24	DA	Transmitter signal element timing (terminal transmitter clock)
25	--	Unassigned

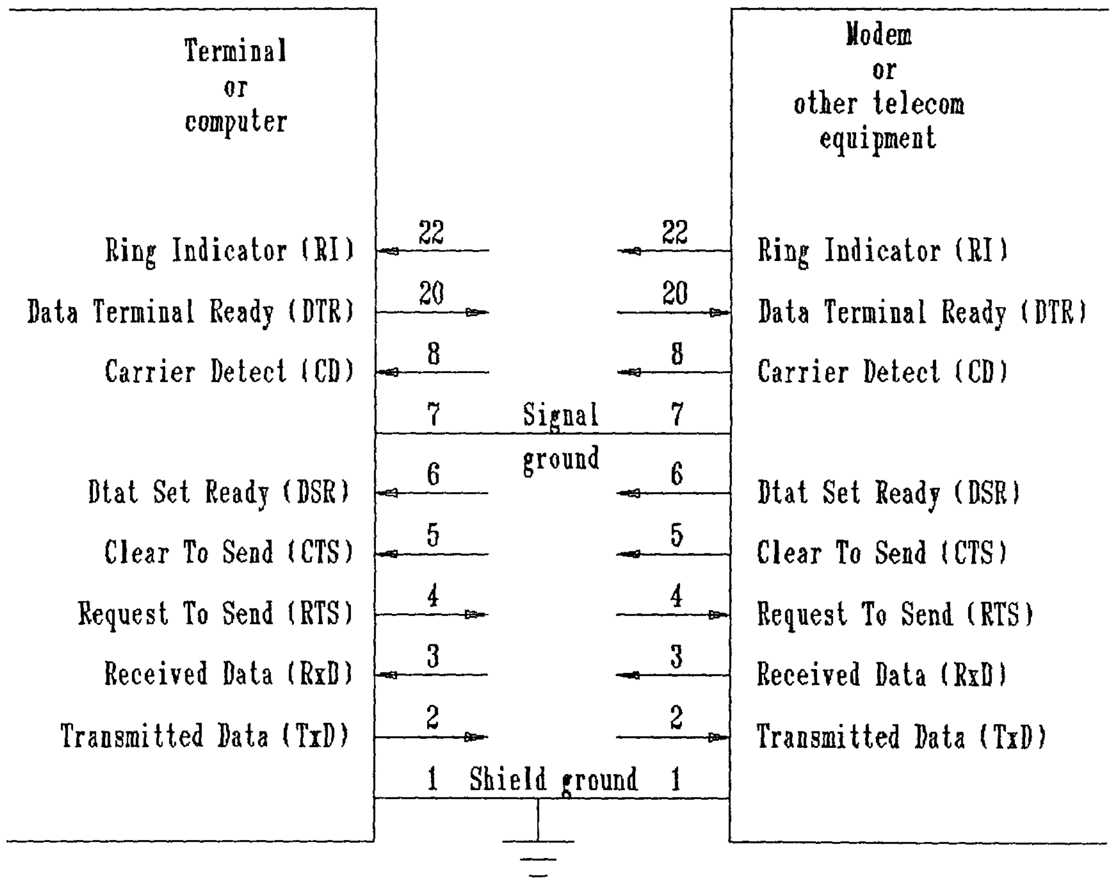


Figure B.1 RS-232C interface with communication equipments

(TxD) and Received Data (RxD). These are compatible signals because Transmitted Data is a modem input and a computer/terminal output, the converse applies to Received Data. Other signals reflect the telephone protocol of the modem. Specifically, Request To Send (RTS) and Clear To Send (CTS) relate to the characteristics of half-duplex telephone lines. Such lines are capable of bi-directional traffic, but they can send in only one direction at a time. The terminal signals a modem with Request To Send when it has characters to transmit, but these characters have to be queued until the modem changes from a receive to a transmit mode. When transmission is allowed, the Clear To Send signal is returned to the terminal, and the transmission begins. The modem can also turn off Clear To Send if the modem moves into the receive mode from transmit mode.

The Clear To Send and Request To Send lines are held to a constant voltage when the telephone connection is full-duplex - that is, when transmit and receive functions can occur simultaneously on independent wires.

Two READY signals are introduced in the RS-232C standard. Data Set Ready (DSR) signifies that the modem is operational, and Data Terminal Ready (DTR) is the corresponding signal for computers and terminals. These signals are sometimes connected to the power supply and become asserted when the device is powered on. The Data Set

Ready signal for a modem indicates more than just a power on condition, the modem is actually connected to a communications line, and is not in a test mode or a disconnected state. The READY signals are then passed across the cable link so that the equipment on the opposite end of the cable can sense the condition.

Carrier Detect (CD) and Ring Indicator (RI) are related to telephone functions. The Ring Indicator is asserted during the period that a ringing tone is present on the communications line. When ringing is sensed at a computer I/O port, the port should post an interrupt, at which point the computer can initiate a connection to the caller. Carrier Detect is a signal that indicates that a remote connection is currently active. If the connection should break for any reason, the Carrier Detect signal is lost, and this too should cause an interrupt at a computer I/O port.

Table B.2 gives the sequence of logic signals needed to establish a connection, transmit data, and terminate the connection. The steps involved in this process are described briefly below.

1. When the computer is ready to accept a call, it sets the Data Terminal Ready signal (DTR) to 1.

Step	Terminal	Interface signals	Data set B (Modem B)	Data set A (Modem A)	Interface signals	Computer
1				Enable answering	CD	$\leftarrow$ 1
2	Dialed digits $\rightarrow$			1 $\rightarrow$ Goes off-hook 1 $\rightarrow$	CE CC	
3		CF	$\leftarrow$ 1	$\leftarrow$ 2225 Hz	CA	$\leftarrow$ 1
4	Push button (1 $\rightarrow$)	CA CB CC	1275 Hz $\rightarrow$ $\leftarrow$ 1 $\leftarrow$ 1	1 $\rightarrow$ 1 $\rightarrow$	CB CF	
5	Output data $\leftarrow$ Input data $\rightarrow$	BB BA	$\leftarrow$ Data 1275-1075 Hz $\rightarrow$	$\leftarrow$ 2225-2025 Hz Data $\rightarrow$	BA BB	$\leftarrow$ Output Data $\rightarrow$ Input Data
6		CF	$\leftarrow$ 0	Drop 2225 Hz Disconnect	CA CD	$\leftarrow$ 0 $\leftarrow$ 0
7	0 $\rightarrow$	CA CB CC	Drop 1275 $\leftarrow$ 0 $\leftarrow$ 0	0 $\rightarrow$ 0 $\rightarrow$ 0 $\rightarrow$	CF CC CB	
8	Terminale connection				CD	$\leftarrow$ 1

Table B.2 The RS-232C standard signaling sequence

2. Data set A (modem A) monitors the telephone line, and when it detects the ringing current, indicating an incoming call, it signals the computer by setting the Ring Indicator (RI) to 1. If DTR = 1 at the time the ringing current is detected, the modem automatically answers the call by going off-hook. Then, it sets the Data Set Ready (DSR) signal to 1.

3. The computer instructs modem A to start transmitting the frequency representing a mark condition (2225 Hz) by setting Request To Send (RTS) to 1. When this is accomplished, modem A responds by setting Clear To Send (CTS) to 1. The detection of the mark frequency at modem B causes it to set the received line signal Carrier Detect (CD) to 1, and turn on a front panel indicator light.

4. The user responds by pressing a button on the front panel of the modem, which is equivalent to setting Request To Send (RTS) to 1, causing transmission of the 1275-Hz signal. Modem B then sets Clear To Send (CTS) and Data Set Ready (DSR) to 1. When modem A detects the 1275-Hz frequency, it sets Carrier Detect (CD) to 1.

5. A full-duplex link is now established between the computer and the remote terminal. The computer can transfer data to and from the remote terminal in the same way as in the case of local terminals. Interface pins TxD

(Transmitted Data) and RxD (Received Data) are used for this purpose, while all other signals in the interface remain unchanged.

6. When the user signs off, the computer sets the Request To Send and Data Terminal Ready (RTS and DTR) signals to 0, causing modem A to drop the mark condition and disconnect from the line. Signals CTS, CD, and DSR are set to 0 by modem A. When modem B senses the disappearance of the mark condition on the line, it sets the received line signal CD to 0.

7. Modem B responds by removing its mark frequency from the line and setting CTS and DSR to 0. The user terminates the connection by going on-hook.

8. The computer sets Data Terminal Ready (DTR) to 1, in preparation for a new call.

Fig. B.2 shows how to connect terminals/computers directly without the use of modems. Note that the Transmitted Data and Received Data lines are crossed so that the devices can, at least, transmit and receive properly. Given the full-duplex nature of the hard-wired connection, Request To Send and Clear To Send no longer serve a modem-like function. Hence, Request To Send is folded back as Clear To Send, and a transmission request is thereby always

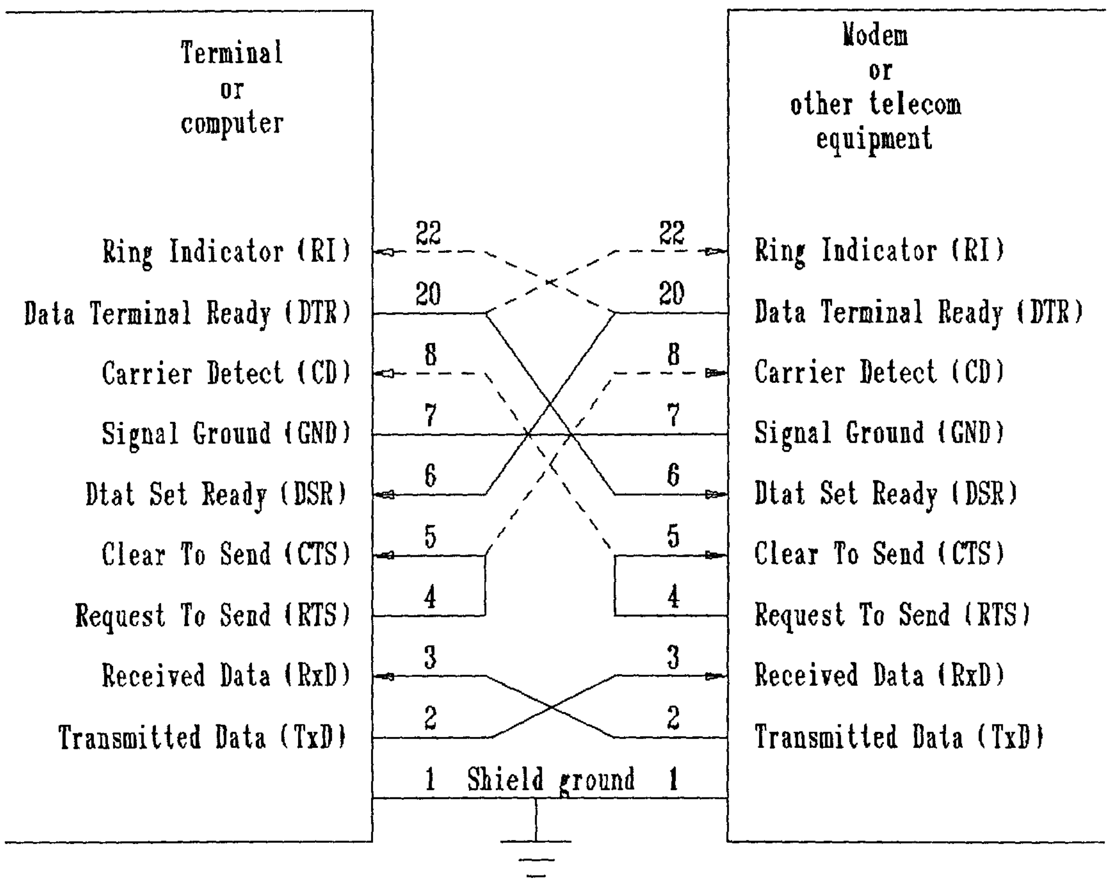


Figure B.2 RS-232C, terminal/computer to terminal/computer

granted. The dotted line shows this signal is used as Carrier Detect at the other end because the presence of Request To Send is functionally similar to the detection of a carrier in a communications channel. The last set of signals, Data Set Ready and Data Terminal Ready are crossed so that each end of the link can detect the presence of a ready condition on the other end. Note the dotted line carries the READY signal to the Ring Indicator input. Dotted lines in this drawing signify connections that are sometimes made in practice, but are less common than the connections as represented by solid lines which cross couple or feed back pairs of signals.

APPENDIX C

SUMMARY OF HP RS-232C SERIAL STATUS AND CONTROL REGISTERS

APPENDIX C

SUMMARY OF HP RS-232C SERIAL STATUS AND CONTROL REGISTERS

STATUS Register 0	Card Identification Value returned: 2 indicates a 96626 (if 130 is returned, the Remote jumper wire has been removed from the interface card); 66 indicates a 98644 (194 if the Remote jumper has been removed).
CONTROL Register 0	Interface Reset Any value from 1 thru 255 resets the card. Execution is immediate; any data transfers in process are aborted and any buffered data is destroyed. A value of 0 causes no action.
STATUS Register 1	Interrupt Status Bit 7 set: Interface hardware interrupt to CPU enabled. Bit 6 set: Card is requesting interrupt service. Bit 5&4: 00 Interrupt Level 3 01 Interrupt Level 4 10 Interrupt Level 5

	11 Interrupt Level 6
	Bit 3 thru 0 not used.
CONTROL Register 1	Transmit BREAK Any non-zero value sends a 400 millisecond BREAK on the serial line.
STATUS Register 2	Interface Activity Status Bit 7 thru 3 are used. Bit 2 set: Handshake in progress. This occurs only during multi-line function calls. Bit 1 set: Firmware interrupts enabled (ENABLE INTR active for this select code). Bit 0: Reserved for future use.
STATUS Register 3	Current Baud Rate Returns one of the values listed under CONTROL Register 3.

CONTROL Register 3	Set new Baud Rate Use any one of the following values:

50	150	1200	4800
75	200	1800	7200
110	300	2400	9600
134.5 (or 134)	600	3600	19200

STATUS Register 4	Current Character Format See CONTROL Register 4 for function of individual bits.

CONTROL Register 4 Set New Character Format

Character Format and Parity Settings

Parity Sense (Switches 5&4)	Parity Enable (Switch 3)	Stop Bits (Switch 2)	Character Length (Switches 1&0)
00 ODD parity	0 Disabled	0 1 stop bit	00 5 bits/char
01 EVEN parity	1 Enabled	1 1.5 stop bits (if 5 bits/char) or 2 stop bits (if 6, 7, or 8 bits/char).	01 6 bits/char
10 Always ONE			10 7 bits/char
11 Always ZERO			11 8 bits/char

Bits 7 and 6 are reserved for future use.

STATUS Register 5 Current Status of Modem Control Lines
Returns CURRENT line state values. See CONTROL Register 5 for function of each bit.

CONTROL Register 5 Set Modem Control Line States
Sets Modem Control lines or interface state as follows:

	Bit 4 set: Enables loopback mode for diagnostic tests.
	Bit 3 set: Set Secondary Request-to-Send modem line to active state.
	Bit 2 set: Set Data Rate Select modem line to active state.
	Bit 1 set: Force Request-to-Send modem line to fixed active state.
	Bit 1 clear: Toggle RTS line as in normal OUTPUT operations.
	Bit 0 set: Force Data Terminal Ready modem line to fixed active state.
	Bit 0 clear: Toggle DTR line as in normal OUTPUT and ENTER operations.
STATUS Register 6	Data In Reads character from input buffer. Buffer contents is not destroyed, but bit 0 of STATUS Register 10 is cleared.
CONTROL Register 6	Data Out Sends character to transmitter holding register. This register is sometimes used to transmit protocol control characters or other characters without using OUTPUT statements. Modem control lines are not affected.

STATUS Register 7 Optional Receiver/Driver Status

Returns current value of optional circuit drivers or receivers as follows:

Bit 3: Optional Circuit Driver 3 (OCD3).

Bit 2: Optional Circuit Driver 4 (OCD4).

Bit 1: Optional Circuit Receiver 2 (OCR2).

Bit 0: Optional Circuit Receiver 3 (OCR3).

Other bits are not used (always 0).

CONTROL Register 7 Set New Optional Drive States

Sets (bit=1) or clears (bit=0) optional circuit drivers as follows:

Bit 3: Optional Circuit Driver 3 (OCD3).

Bit 2: Optional Circuit Driver 4 (OCD4).

Other bits are not used.

STATUS Register 8 Current Interrupt Enable Mask

Returns value of interrupt mask associated with most recent ENABLE INTR statement. Bit functions are as follows:

Bit 3: Enable interrupt on modem line change. STATUS Register 11 shows which modem line has changed.

Bit 2: Enable interrupt on UART status error. This bit is used to trap ERROR 167 caused by UART error conditions. STATUS REgister 10, bits 4 thru 1, show cause of error.

Bit 1: Enable interrupt when Transmitter Holding Register is empty.

Bit 0: Enable interrupt when Receiver Buffer is full.

STATUS Register 9	Cause of Current Interrupt

Returns cause of interrupt as follows:

Bits 2&1: Return cause of interrupt

11=UART error (BREAK, parity, framing, or overrun error). See STATUS Register 10.

10=Receiver Buffer full. Cleared by STATUS to Register 6.

01=Transmitter HOlding Register empty. Cleared by CONTROL Register 6 or STATUS to Register 9.

00=Interrupt caused by change in modem status line(s). See STATUS Register 11.

Bit 0: Set when no active interrupt requests from UART are pending. Clear until all pending interrupts have been serviced.

STATUS Register 10 UART Status

Bit set indicates UART status or detected error as follows:

Bit 7: Not used.

Bit 6: Transmit Shift Register empty.

Bit 5: Transmit Holding Register empty.

Bit 4: Break received.

Bit 3: Framing error detected.

Bit 2: Parity error detected.

Bit 1: Receive Buffer Overrun error.

Bit 0: Receiver Buffer full.

STATUS Register 11 Modem Status

Bit set indicates that the specified modem line or condition active.

Bit 7: Data Carrier Detect (DCD) modem line active.

Bit 6: Ring Indicator (RI) modem line active.

Bit 5: Data Set Ready (DSR) modem line active.

Bit 4: Clear-to-Send (CTS) modem line active.

Bit 3: Change in DCD line state detected.

Bit 2: RI modem line changed from true to false.

Bit 1: Change in DSR line state detected.

Bit 0: Change in CTS line state detected.

STATUS Register 12

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Carrier Detect Disable	0	Data Set Ready Disable	Clear to Send Disable	0	0	0	0
Val=128	Val=64	Val=0	Val=16	Val=8	Val=4	Val=2	Val=1

CONTROL Register 12 Modem Handshake Control

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Carrier Detect Disable	Not Used	Data Set Ready Disable	Clear to Send Disable	Not Used			
Val=128	Val=64	Val=0	Val=16	Val=8	Val=4	Val=2	Val=1

Interrupt Enable Register (ENABLE INTR)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used				Modem Status Change	Receiver Line Status	Trans-mitter Holding Regis-ter Empty	Receiver Buffer Full
Val=128	Val=64	Val=0	Val=16	Val=8	Val=4	Val=2	Val=1

STATUS Register 13	Read 98644 "SCRATCH A default" baud rate Returns the baud rate that will be restored whenever SCRATCH A is executed (same bit-definitions as STATUS register 3).
CONTROL Register 13	Set 98644 "SCRATCH A default" baud rate Sets both the "current"and the "default" baud rate that will be restored whenever SCRATCH A is executed (same bit-definitions as CONTROL register 3). default value in this register is 9600 baud.
STATUS Register 14	Read 98644 "SCRATCH A default" character format Returns the character format parameters that will be restored whenever SCRATCH A is executed (same bit-definitions as STATUS register 4).
CONTROL Register 14	Set 98644 "SCRATCH A default" character format Sets the character format parameters that will be restored whenever SCRATCH A is executed (same bit-definitions as

CONTROL register 4). Default value in this register specifies a character format of 8 bits/character, 1 stop bit, and parity disabled.

APPENDIX D

REFERENCE FILES FOR DEMONSTRATION PART

```
fmlab5% cat pp_ccp_test

-- PROCESS_PLAN --


-- HEADER_SECTION --
        PLAN_ID          := pp_ccp_test;
        PLAN_VERSION     := 1;
        PLAN_TYPE        := PP.CCP#1;
        PALN_NAME        := RECTANG_SAMPLE;
-- END_HEADER_SECTION --


-- PARAMETERS_SECTION --
-- END_PARAMETERS_SECTION --


-- REQUIREMENTS_SECTION --
        << 1 >> CELL_ELEMENT
               ( ELEMENT_ID     => VMC          );
        << 2 >> CELL_ELEMENT
               ( ELEMENT_ID     => VISE         );
        << 3 >> CELL_ELEMENT
               ( ELEMENT_ID     => GRIPPER      );
        << 4 >> CELL_ELEMENT
               ( ELEMENT_ID     => ROBOT        );
-- END_REQUIREMENTS_SECTION --


-- PROCEDURE_SECTION --
        << 1 >> INIT
        << 2 >> LOAD_PART
                 ( PART_TYPE       => RECTANGULAR  ,
                   PART_SIZE_L     => 6.0          ,
                   PART_SIZE_W     => 3.0          ,
                   PART_SIZE_D     => 1.0          ,
                   PART_MATERIAL   => AL           ,
                   PREC_STEPS      => ()           ,
                   TIME            => 0000:00:01:00 );
        << 3 >> MACHINE_PART
                 ( NCFILE_NAME     => easydemo.nc  ,
                   PREC_STEPS      => (2)          ,
                   TIME            => 0000:00:30:00 );
        << 4 >> UNLOAD_PART
                 ( PART_TYPE       => RECTANGULAR  ,
                   PART_SIZE_L     => 6.0          ,
                   PART_SIZE_W     => 3.0          ,
                   PART_SIZE_D     => 1.0          ,
                   PART_MATERIAL   => AL           ,
                   PREC_STEPS      => (3)          ,
                   TIME            => 0000:00:01:00 );
        << 5 >> CLOSE
-- END_PROCEDURE_SECTION --


-- END_PROCESS_PLAN --
```

```
fmlab4% cat /usr/unger/vws2/design/nc/chendemo_nc
%O0001
N0001 ( HEADER PROG =  CHENDEMO_NC  PROG_NAME=
CHEN_DEMO_PART )
N0002 G91 G28 Z+0.0
N0003 G91 G28 X+0.0 Y+0.0
N0004 G52
N0005 G92 X+0.0 Y+0.0 Z+0.0
N0006 G10 P1 Q2 X-11.2 Y-9.0 Z+0.0
N0007 G0 G90 G54 X+0.0 Y+0.0
N0008 ( 0.1 BY 1.0 BY 1.0 POCKET )
N0009 ( CHANGING TOOL TO  0.375 INCH DIAMETER  END_MILL )
N0010 M9
N0011 G0
N0012 G53 Z+0.0
N0013 T3 M6
N0014 G90 G0 S1528 M3
N0015 G43 H3 Z+10.0
N0016 M8
N0017 X+2.7 Y+0.7
N0018 G0 Z+0.1
N0019 G1 Z-0.1 F3.82 M8
N0020 G1 X+2.7 Y+0.7
N0021 X+2.8875 Y+0.5125
N0022 Y+0.8875
N0023 X+2.5125
N0024 Y+0.5125
N0025 X+2.8875
N0026 G1 X+3.0 Y+0.3975
N0027 G3 X+3.0025 Y+0.4 R+0.0025
N0028 G1 Y+1.0
N0029 G3 X+3.0 Y+1.0025 R+0.0025
N0030 G1 X+2.4
N0031 G3 X+2.3975 Y+1.0 R+0.0025
N0032 G1 Y+0.4
N0033 G3 X+2.4 Y+0.3975 R+0.0025
N0034 G1 X+3.0
N0035 G0 Z+1.0
N0036 X+3.0 Y+0.3875
N0037 Z+0.1
N0038 G1 Z-0.1 F3.82
N0039 F7.64
N0040 G3 X+3.0125 Y+0.4 R+0.0125
N0041 G1 Y+1.0
N0042 G3 X+3.0 Y+1.0125 R+0.0125
N0043 G1 X+2.4
N0044 G3 X+2.3875 Y+1.0 R+0.0125
N0045 G1 Y+0.4
N0046 G3 X+2.4 Y+0.3875 R+0.0125
N0047 G1 X+3.0
N0048 ( 0.3 BY 0.5 BY 0.8 POCKET )
N0049 G90 G0 Z+1.0
N0050 X+4.5 Y+0.85
```

```
N0051 G0 Z+0.1
N0052 G1 Z-0.1875 F3.82 M8
N0053 G1 X+4.5 Y+0.55
N0054 G1 X+4.55 Y+0.4975
N0055 G3 X+4.5525 Y+0.5 R+0.0025
N0056 G1 Y+0.9
N0057 G3 X+4.55 Y+0.9025 R+0.0025
N0058 G1 X+4.45
N0059 G3 X+4.4475 Y+0.9 R+0.0025
N0060 G1 Y+0.5
N0061 G3 X+4.45 Y+0.4975 R+0.0025
N0062 G1 X+4.55
N0063 G0 Z+1.0
N0064 X+4.5 Y+0.85
N0065 G0 Z+0.1
N0066 G1 Z-0.3 F3.82 M8
N0067 G1 X+4.5 Y+0.55
N0068 G1 X+4.55 Y+0.4975
N0069 G3 X+4.5525 Y+0.5 R+0.0025
N0070 G1 Y+0.9
N0071 G3 X+4.55 Y+0.9025 R+0.0025
N0072 G1 X+4.45
N0073 G3 X+4.4475 Y+0.9 R+0.0025
N0074 G1 Y+0.5
N0075 G3 X+4.45 Y+0.4975 R+0.0025
N0076 G1 X+4.55
N0077 G0 Z+1.0
N0078 X+4.55 Y+0.4875
N0079 Z+0.1
N0080 G1 Z-0.3 F3.82
N0081 F7.64
N0082 G3 X+4.5625 Y+0.5 R+0.0125
N0083 G1 Y+0.9
N0084 G3 X+4.55 Y+0.9125 R+0.0125
N0085 G1 X+4.45
N0086 G3 X+4.4375 Y+0.9 R+0.0125
N0087 G1 Y+0.5
N0088 G3 X+4.45 Y+0.4875 R+0.0125
N0089 G1 X+4.55
N0090 ( 0.1 BY 2.5 BY 1.0 POCKET )
N0091 G90 G0 Z+1.0
N0092 X+3.75 Y+0.7
N0093 G0 Z+0.1
N0094 G1 Z-0.1 F3.82 M8
N0095 G1 X+5.25 Y+0.7
N0096 X+5.4375 Y+0.5125
N0097 Y+0.8875
N0098 X+3.5625
N0099 Y+0.5125
N0100 X+5.4375
N0101 G1 X+5.55 Y+0.3975
N0102 G3 X+5.5525 Y+0.4 R+0.0025
N0103 G1 Y+1.0
```

```
N0104 G3 X+5.55 Y+1.0025 R+0.0025
N0105 G1 X+3.45
N0106 G3 X+3.4475 Y+1.0 R+0.0025
N0107 G1 Y+0.4
N0108 G3 X+3.45 Y+0.3975 R+0.0025
N0109 G1 X+5.55
N0110 G0 Z+1.0
N0111 X+5.55 Y+0.3875
N0112 Z+0.1
N0113 G1 Z-0.1 F3.82
N0114 F7.64
N0115 G3 X+5.5625 Y+0.4 R+0.0125
N0116 G1 Y+1.0
N0117 G3 X+5.55 Y+1.0125 R+0.0125
N0118 G1 X+3.45
N0119 G3 X+3.4375 Y+1.0 R+0.0125
N0120 G1 Y+0.4
N0121 G3 X+3.45 Y+0.3875 R+0.0125
N0122 G1 X+5.55
N0123 (  0.1  BY  1.026  BY  0.7869  CONTOUR_POCKET )
N0124 ( CAUTION UNTESTED CONTOUR_POCKET_NC #1 )
N0125 ( CHANGING TOOL TO  0.125 INCH DIAMETER  END_MILL )
N0126 M9
N0127 G0
N0128 G53 Z+0.0
N0129 T1 M6
N0130 G90 G0 S4584 M3
N0131 G43 H1 Z+10.0
N0132 M8
N0133 X+0.6313 Y+1.0246
N0134 Z+0.1
N0135 G1 Z-0.0625 F4.5
N0136 F9.0
N0137 G1 X+0.8325
N0138 G1 X+0.9262 Y+0.9621
N0139 G1 X+0.5743
N0140 G3 X+0.5595 Y+0.8996 R+0.1375
N0141 G1 X+1.4405
N0142 G2 X+1.4253 Y+0.8371 R+0.1375
N0143 G1 X+0.5747
N0144 G3 X+0.5998 Y+0.8028 R+0.1375
N0145 G1 X+0.628 Y+0.7746
N0146 G1 X+1.372
N0147 G1 X+1.3095 Y+0.7121
N0148 G1 X+0.6905
N0149 G1 X+0.753 Y+0.6496
N0150 G1 X+1.247
N0151 G1 X+1.1845 Y+0.5871
N0152 G1 X+0.8155
N0153 G1 X+0.878 Y+0.5246
N0154 G1 X+1.122
N0155 G1 X+1.0595 Y+0.4621
N0156 G1 X+0.9405
```

```
N0157 G0 Z+1.0
N0158 X+1.1675 Y+1.0246
N0159 Z+0.1
N0160 G1 Z-0.0625 F4.5
N0161 F9.0
N0162 G1 X+1.3687
N0163 G2 X+1.3937 Y+1.0038 R+0.1775
N0164 G1 X+1.4002 Y+0.9972
N0165 G2 X+1.4257 Y+0.9621 R+0.1375
N0166 G1 X+1.0738
N0167 G0 Z+1.0
N0168 X+0.9806 Y+0.422
N0169 Z+0.1
N0170 G1 Z-0.0625 F4.5
N0171 F9.0
N0172 G3 X+1.0194 Y+0.422 R+0.0275
N0173 G1 X+1.4002 Y+0.8028
N0174 G3 X+1.4002 Y+0.9972 R+0.1375
N0175 G1 X+1.3937 Y+1.0038
N0176 G3 X+1.1697 Y+1.026 R+0.1775
N0177 G1 X+1.0402 Y+0.9397
N0178 G2 X+0.9598 Y+0.9397 R+0.0725
N0179 G1 X+0.8303 Y+1.026
N0180 G3 X+0.6063 Y+1.0038 R+0.1775
N0181 G1 X+0.5998 Y+0.9972
N0182 G3 X+0.5998 Y+0.8028 R+0.1375
N0183 G1 X+0.9806 Y+0.422
N0184 G0 Z+1.0
N0185 X+0.6313 Y+1.0246
N0186 Z+0.1
N0187 G1 Z-0.1 F4.5
N0188 F9.0
N0189 G1 X+0.8325
N0190 G1 X+0.9262 Y+0.9621
N0191 G1 X+0.5743
N0192 G3 X+0.5595 Y+0.8996 R+0.1375
N0193 G1 X+1.4405
N0194 G2 X+1.4253 Y+0.8371 R+0.1375
N0195 G1 X+0.5747
N0196 G3 X+0.5998 Y+0.8028 R+0.1375
N0197 G1 X+0.628 Y+0.7746
N0198 G1 X+1.372
N0199 G1 X+1.3095 Y+0.7121
N0200 G1 X+0.6905
N0201 G1 X+0.753 Y+0.6496
N0202 G1 X+1.247
N0203 G1 X+1.1845 Y+0.5871
N0204 G1 X+0.8155
N0205 G1 X+0.878 Y+0.5246
N0206 G1 X+1.122
N0207 G1 X+1.0595 Y+0.4621
N0208 G1 X+0.9405
N0209 G0 Z+1.0
```

```
N0210 X+1.1675 Y+1.0246
N0211 Z+0.1
N0212 G1 Z-0.1 F4.5
N0213 F9.0
N0214 G1 X+1.3687
N0215 G2 X+1.3937 Y+1.0038 R+0.1775
N0216 G1 X+1.4002 Y+0.9972
N0217 G2 X+1.4257 Y+0.9621 R+0.1375
N0218 G1 X+1.0738
N0219 G0 Z+1.0
N0220 X+0.9806 Y+0.422
N0221 Z+0.1
N0222 G1 Z-0.1 F4.5
N0223 F9.0
N0224 G3 X+1.0194 Y+0.422 R+0.0275
N0225 G1 X+1.4002 Y+0.8028
N0226 G3 X+1.4002 Y+0.9972 R+0.1375
N0227 G1 X+1.3937 Y+1.0038
N0228 G3 X+1.1697 Y+1.026 R+0.1775
N0229 G1 X+1.0402 Y+0.9397
N0230 G2 X+0.9598 Y+0.9397 R+0.0725
N0231 G1 X+0.8303 Y+1.026
N0232 G3 X+0.6063 Y+1.0038 R+0.1775
N0233 G1 X+0.5998 Y+0.9972
N0234 G3 X+0.5998 Y+0.8028 R+0.1375
N0235 G1 X+0.9806 Y+0.422
N0236 G0 Z+1.0
N0237 X+0.9735 Y+0.4149
N0238 Z+0.1
N0239 G1 Z-0.1 F4.5
N0240 F9.0
N0241 G3 X+1.0265 Y+0.4149 R+0.0375
N0242 G1 X+1.4073 Y+0.7957
N0243 G3 X+1.4073 Y+1.0043 R+0.1475
N0244 G1 X+1.4007 Y+1.0109
N0245 G3 X+1.1641 Y+1.0343 R+0.1875
N0246 G1 X+1.0347 Y+0.948
N0247 G2 X+0.9653 Y+0.948 R+0.0625
N0248 G1 X+0.8359 Y+1.0343
N0249 G3 X+0.5993 Y+1.0109 R+0.1875
N0250 G1 X+0.5927 Y+1.0043
N0251 G3 X+0.5927 Y+0.7957 R+0.1475
N0252 G1 X+0.9735 Y+0.4149
N0253 (  0.1 BY  5.525  BY  1.325 CONTOUR_GROOVE OF  0.125
WIDTH )
N0254 G90 G0 Z+1.0
N0255 X+0.55 Y+2.7
N0256 Z+0.1
N0257 G1 Z-0.0625 F4.5
N0258 F9.0
N0259 G3 X+0.3 Y+2.45 R+0.25
N0260 G1 X+0.3 Y+1.75
N0261 G3 X+0.55 Y+1.5 R+0.25
```

```
N0262 G1 X+5.45 Y+1.5
N0263 G3 X+5.7 Y+1.75 R+0.25
N0264 G1 X+5.7 Y+2.45
N0265 G3 X+5.45 Y+2.7 R+0.25
N0266 G1 X+0.55 Y+2.7
N0267 G0 Z+1.0
N0268 X+0.55 Y+2.7
N0269 Z+0.1
N0270 G1 Z-0.1 F4.5
N0271 F9.0
N0272 G3 X+0.3 Y+2.45 R+0.25
N0273 G1 X+0.3 Y+1.75
N0274 G3 X+0.55 Y+1.5 R+0.25
N0275 G1 X+5.45 Y+1.5
N0276 G3 X+5.7 Y+1.75 R+0.25
N0277 G1 X+5.7 Y+2.45
N0278 G3 X+5.45 Y+2.7 R+0.25
N0279 G1 X+0.55 Y+2.7
N0280 ( 0.1 DEEP TEXT FMLAB )
N0281 ( CHANGING TOOL TO  0.125 INCH DIAMETER
BALL_NOSED_END_MILL )
N0282 M9
N0283 G0
N0284 G53 Z+0.0
N0285 T29 M6
N0286 G90 G0 M3
N0287 G43 H29 Z+10.0
N0288 M8
N0289 X+1.675 Y+1.85
N0290 Z+0.1
N0291 G1 Z-0.0625 F4.5 ( F )
N0292 G1 X+1.675 Y+2.35 F9.0
N0293 X+2.0083
N0294 G0 Z+1.0
N0295 X+1.675 Y+2.1417
N0296 Z+0.1
N0297 G1 Z-0.0625 F4.5
N0298 X+1.925 F9.0
N0299 G0 Z+1.0
N0300 X+1.675 Y+1.85
N0301 Z+0.1
N0302 G1 Z-0.1 F4.5
N0303 G1 X+1.675 Y+2.35 F9.0
N0304 X+2.0083
N0305 G0 Z+1.0
N0306 X+1.675 Y+2.1417
N0307 Z+0.1
N0308 G1 Z-0.1 F4.5
N0309 X+1.925 F9.0
N0310 G0 Z+1.0
N0311 X+2.175 Y+1.85
N0312 Z+0.1
N0313 G1 Z-0.0625 F4.5 ( M )
```

```
N0314 G1 X+2.175 Y+2.35 F9.0
N0315 X+2.425 Y+1.9333
N0316 X+2.675 Y+2.35
N0317 Y+1.85
N0318 G0 Z+1.0
N0319 X+2.175 Y+1.85
N0320 Z+0.1
N0321 G1 Z-0.1 F4.5
N0322 G1 X+2.175 Y+2.35 F9.0
N0323 X+2.425 Y+1.9333
N0324 X+2.675 Y+2.35
N0325 Y+1.85
N0326 G0 Z+1.0
N0327 X+3.175 Y+1.85
N0328 Z+0.1
N0329 G1 Z-0.0625 F4.5 ( L )
N0330 G1 X+2.8417 Y+1.85 F9.0
N0331 Y+2.35
N0332 G0 Z+1.0
N0333 X+3.175 Y+1.85
N0334 Z+0.1
N0335 G1 Z-0.1 F4.5
N0336 G1 X+2.8417 Y+1.85 F9.0
N0337 Y+2.35
N0338 G0 Z+1.0
N0339 X+3.3417 Y+1.85
N0340 Z+0.1
N0341 G1 Z-0.0625 F4.5 ( A )
N0342 G1 X+3.5917 Y+2.35 F9.0
N0343 X+3.8417 Y+1.85
N0344 G0 Z+1.0
N0345 X+3.7583 Y+2.0167
N0346 Z+0.1
N0347 G1 Z-0.0625 F4.5
N0348 X+3.425 F9.0
N0349 G0 Z+1.0
N0350 X+3.3417 Y+1.85
N0351 Z+0.1
N0352 G1 Z-0.1 F4.5
N0353 G1 X+3.5917 Y+2.35 F9.0
N0354 X+3.8417 Y+1.85
N0355 G0 Z+1.0
N0356 X+3.7583 Y+2.0167
N0357 Z+0.1
N0358 G1 Z-0.1 F4.5
N0359 X+3.425 F9.0
N0360 G0 Z+1.0
N0361 X+4.0083 Y+1.85
N0362 Z+0.1
N0363 G1 Z-0.0625 F4.5 ( B )
N0364 G1 X+4.0083 Y+2.35 F9.0
N0365 X+4.2583
N0366 G2 X+4.3417 Y+2.2667 R+0.0833
```

```
N0367 G1 Y+2.1833
N0368 G2 X+4.2583 Y+2.1 R+0.0833
N0369 G1 X+4.0083
N0370 G0 Z+1.0
N0371 X+4.2583 Y+2.1
N0372 Z+0.1
N0373 G1 Z-0.0625 F4.5
N0374 G2 X+4.3417 Y+2.0167 R+0.0833 F9.0
N0375 G1 Y+1.9333
N0376 G2 X+4.2583 Y+1.85 R+0.0833
N0377 G1 X+4.0083
N0378 G0 Z+1.0
N0379 X+4.0083 Y+1.85
N0380 Z+0.1
N0381 G1 Z-0.1 F4.5
N0382 G1 X+4.0083 Y+2.35 F9.0
N0383 X+4.2583
N0384 G2 X+4.3417 Y+2.2667 R+0.0833
N0385 G1 Y+2.1833
N0386 G2 X+4.2583 Y+2.1 R+0.0833
N0387 G1 X+4.0083
N0388 G0 Z+1.0
N0389 X+4.2583 Y+2.1
N0390 Z+0.1
N0391 G1 Z-0.1 F4.5
N0392 G2 X+4.3417 Y+2.0167 R+0.0833 F9.0
N0393 G1 Y+1.9333
N0394 G2 X+4.2583 Y+1.85 R+0.0833
N0395 G1 X+4.0083
N0396 ( 0.1 BY 1.5 BY 0.9 POCKET )
N0397 ( CHANGING TOOL TO  0.375 INCH DIAMETER  END_MILL )
N0398 M9
N0399 G0
N0400 G53 Z+0.0
N0401 T3 M6
N0402 G90 G0 S1528 M3
N0403 G43 H3 Z+10.0
N0404 M8
N0405 X+4.2 Y+0.7
N0406 G0 Z+0.0
N0407 G1 Z-0.2 F3.82 M8
N0408 G1 X+4.8 Y+0.7
N0409 X+4.9875 Y+0.5125
N0410 Y+0.8875
N0411 X+4.0125
N0412 Y+0.5125
N0413 X+4.9875
N0414 G1 X+5.05 Y+0.4475
N0415 G3 X+5.0525 Y+0.45 R+0.0025
N0416 G1 Y+0.95
N0417 G3 X+5.05 Y+0.9525 R+0.0025
N0418 G1 X+3.95
N0419 G3 X+3.9475 Y+0.95 R+0.0025
```

```
N0420 G1 Y+0.45
N0421 G3 X+3.95 Y+0.4475 R+0.0025
N0422 G1 X+5.05
N0423 G0 Z+1.0
N0424 X+5.05 Y+0.4375
N0425 Z+0.0
N0426 G1 Z-0.2 F3.82
N0427 F7.64
N0428 G3 X+5.0625 Y+0.45 R+0.0125
N0429 G1 Y+0.95
N0430 G3 X+5.05 Y+0.9625 R+0.0125
N0431 G1 X+3.95
N0432 G3 X+3.9375 Y+0.95 R+0.0125
N0433 G1 Y+0.45
N0434 G3 X+3.95 Y+0.4375 R+0.0125
N0435 G1 X+5.05
N0436 G0
N0437 M9
N0438 M5
N0439 G53 Z+0.0
N0440 G53 X+0.0 Y+0.0
N0441 M30
N0442 ( END OF PROGRAM )
%
```

```
fmlab18% cat vmc_setup

setup_#   x_position   y_position   z_position     vise_l         vise_h
--------- ------------ ------------ -------------- -------------- -------------
( 1 )     -11.748      -7.081       0.0            6.125          4.500
( 2 )     -12.125      -6.0         0.0            6.125          4.500
```

```
fmlab19% cat ccfile

COM VMC             INITIAL                                              %
COM GRIPPER         OPEN                                                 %
COM VISE            OPEN                                                 %
COM ROBOT           0.0000           160.0000         14.5000            %
COM ROBOT           0.0000           160.0000         2.5000             %
COM GRIPPER         CLOSE                                                %
COM ROBOT           -14.8105         -5.4810          6.7000             %
COM ROBOT           -14.8105         -5.4810          4.7000             %
COM GRIPPER         OPEN                                                 %
COM VISE            CLOSE                                                %
COM ROBOT           -14.8105         -5.4810          6.7000             %
COM ROBOT           0.0000           80.0000          8.0000             %
COM VMC             DOWNLOAD         easydemo.nc                         %
COM VMC             PRE_SET                                              %
COM VMC             EXECUTE                                              %
COM VMC             INITIAL                                              %
COM GRIPPER         OPEN                                                 %
COM ROBOT           -14.8105         -5.4810          6.7000             %
COM ROBOT           -14.8105         -5.4810          4.7000             %
COM GRIPPER         CLOSE                                                %
COM VISE            OPEN                                                 %
COM ROBOT           -14.8105         -5.4810          6.7000             %
COM ROBOT           0.0000           160.0000         14.5000            %
COM ROBOT           0.0000           160.0000         2.6000             %
COM GRIPPER         OPEN                                                 %
COM ROBOT           0.0000           160.0000         14.5000            %
COM ROBOT           0.0000           80.0000          8.0000             %
```

APPENDIX E

PROGRAM LISTINGS

```
fmlab2% cat auto_main2.c


/* ---------------------------------------------------------------------- *
 * This is the main program for Cell Control Program (CCP) which is the    *
 * brain of UMCP VMC (vertival machining cell) cell host.                  *
 *                                                                         *
 * When in manual mode, the CCP take the input from user and perform       *
 * necessary steps to complete the task requested by user. When in auto    *
 * mode, CCP take the command from SCP (system control program) and        *
 * process the process plan file given by SCP, capture required            *
 * information, and manipulate data to form the Cell Command File          *
 * (ccfile). According to the cell command file, the CCP than execute the  *
 * cell commands to perform tasks.                                         *
 * ---------------------------------------------------------------------- */



#include <stdio.h>
#define  READMODE    0444
#define  WRITEMODE   0222
#define  ON          1
#define  OFF         0


char *ecp,*scp,*comin,*comout,*stabuf[6];


main()
{
    int ccpmode,atoi(),manmode,quit_ccp_flag,quit_man_flag,quit_auto_flag;
    /* ccpmode - indicate the current ccp mode under the main menu.
       manmode - indicate the current ccp mode under the manual mode.
       quit_*_flag - flags of quit status. */

    int openf1,i,j;
    char temp[30],*ecpsta(),*sta,*str,*rdy,*dwn,*ccfn,*ncfile,*mail,*tempf;
    char eecp[64],*e,*path;
    FILE *fp,*ppfp,*ccfp,*ccfile;


/* print the header */
printf("\n\n\n");
printf("%s\n%s\n%s\n%s\n%s\n%s\n%s\n",
   "*********************************************************************",
   "*                                                                   *",
   "*  Welcome to the Cell Control Program (CCP) of the UMCP flexible  *",
   "*  manufacturing Cell. Please follow the instructions given by     *",
   "*  the computer to perform desired tasks.                          *",
   "*                                                                   *",
   "*********************************************************************");
printf("\n\n");
```

```
/* initialize and space buffers */
for (i=0;i<6;++i){
    stabuf[i]="                                                                %";
}
scp="                                                                %";
ecp="                                                                %";
comin="                                                                %";
comout="                                                                %";
sta="                                                                %";
str="                                                                %";
path="                                         ";


/* loop main menu */
quit_ccp_flag = OFF;
do{
    /* print the ccp main menu and select entry mode */
    printf("\n%s%s%s%s","1. manual mode   ",
                        "2. auto mode     ",
                        "3. help mode     ",
                        "4. quit ccp      ");
    do{
        printf("\n%s","ccp=>");
        scanf("%s",temp);
        ccpmode=atoi(temp);
        if ((ccpmode <1) || (ccpmode > 4)) {
            printf("%s","input error. valid number 1 .. 4");
        }
        else
           break;
    }
    while (1);

    /* branch the program based on the ccp mode */
    switch (ccpmode){
    case 1 :       /* case 1 is manual mode */
        /* loop the ccp/manual menu */
        quit_man_flag = OFF;
        do{
            /* print the ccp/manual menu and select the entry mode */
            printf("\n\n%s%s%s%s","1. status request    ",
                                  "2. service request    ",
                                  "3. help    ",
                                  "4. quit manual mode");
            do{
                printf("\n%s","ccp/manual=>");
                scanf("%s",temp);
                manmode=atoi(temp);
                if ((manmode <1) || (manmode > 4)) {
                    printf("%s","input error. valid number 1 .. 4");
                }
                else
                    break;
        }
        while (1);

        /* branch the manual mode based on the ccp/manual mode */
        switch (manmode){
        case 1:    /* status request */
            printf("\n%s","I am about to enter stareq.c");
            stareq();
            printf("\nI am in main program. ecp[65]=\n%s",ecp);
            break;
        case 2:    /* service request */
            srvreq();
            break;
        case 3 :   /* ccp/manual help mode */
            /* manhelp(); */
            break;
        case 4 :   /* quit manual mode */
            quit_man_flag = ON;
        }
    }
    while (quit_man_flag == OFF);
    break;
case 2 :       /* case 2 is auto mode */
    printf("\nI am in automode");
    rdy="READY ";
    dwn="DOWNLOAD";
    ncfile="                ";
    /* ncfile="/usr/unger/vws2/design/nc/chendemo_nc"; */
    /* ncfile="/usr/unger/vws2/design/nc/fmcdemo_nc"; */
    /* ncfile="/usr/unger/vws2/design/nc/nsfdemo_nc"; */
    /* ncfile="/usr/unger/vws2/design/nc/magrab1_wax_mould"; */
    quit_auto_flag=OFF;

    /* ----------------- check cell status --------------------- */
    str="STA ECP     READY                                         %";
    /* copy command to com_out register */
    strcpy(comout,str);
    printf("\ncomout :%s",comout);
    /* copy command to ecp mailbox */
    strcpy(ecp,str);
    /* command execution */
    ecpcom(ecp);
    /* copy the returned message to appropriate status buffer */
    strcpy(stabuf[0],ecp);
    printf("\nvmc_sta :%s",stabuf[0]);
    if (strncmp(stabuf[0]+12,rdy,6)!=0){
        printf("\n%s","ECP is not ready, back to main menu.");
        quit_auto_flag=ON;
        break;
    }

    /* -------------------- cell initialization ---------------------- */
    str="COM ECP     READY INITIAL                                     %";
    /* copy command to com_out register */
```

```
strcpy(comout,str);
printf("\ncomout :%s",comout);
/* copy command to ecp mailbox */
strcpy(ecp,str);
/* command execution */
ecpcom(ecp);
/* copy the returned message to appropriate status buffer */
strcpy(stabuf[0],ecp);
printf("\nvmc_sta :%s",stabuf[0]);
if (strncmp(stabuf[0]+12,rdy,6)!=0){
    printf("\n%s","cell initialization fails, back to main menu.");
    quit_auto_flag=ON;
    break;
}

/* ------------------- loop the service routine -------------------- */
mail="/usr/chen/automation/scpmail";
do{
    /* report the ccp status to scp for engaging with scp */
    if ((fp=fopen(mail,"w"))==NULL){
        printf("\n%s","scpmail can't be opened, back to main menu.");
        quit_auto_flag=ON;
        break;
    }
    printf("\n%s","scpmail is opened and will be closed soon");
    fprintf(fp,"%s","ready");
    fclose(fp);

    /* receive scp command */
    tempf="/usr/chen/automation/tempfile";
    if ((fp=fopen(tempf,"r")) == NULL){
        printf("\n%s",
        "scp command source can't be opened, back to main menu");
        quit_auto_flag=ON;
        break;
    }
    printf("\n%s","tempfile is opened");
    do{
        if (fscanf(fp,"%64c",eecp)==1) {
            for (i=0;i<64;++i) {
                *(scp+i) = eecp[i];
            }
            *(scp+64) = '\0';
            printf("\nscp = %s",scp);
            fclose(fp);
            break;
        }
        rewind(fp);
    }
    while (1);
    strcpy(comin,scp);

    /* extract the path */
    strncpy(str,scp+18,15);
```

```
*(str+15) = '\0';
printf("\npp_file :%s:",str);
for (i=0;i<15;++i){
    if (*(str+i) == ' ') {
        *(str+i) = '\0';
        *(str+15) = ' ';
        break;
    }
}
printf("\npp_file :%s:\n",str);
path="/usr/chen/automation/";
strcat(path,str);
printf("\n%s",path);
strcpy(str,path);
printf("\nexecuting process plan ID :%s.",str);

/* --------- ccfile generation ---------- */
/* open process-plan file with file name found from scp command */
if ((ppfp=fopen(str,"r"))==NULL){
    printf("\n%s","pp_file can't be opened, back to main menu.");
    quit_auto_flag=ON;
    break;
}
printf("\npp_file is opened");
/* open cell-command file to write */
ccfn="/usr/chen/automation/ccfile";
if ((ccfp=fopen(ccfn,"w"))==NULL){
    printf("\n%s","ccf_file can't be opened, back to main menu.");
    quit_auto_flag=ON;
    break;
}
printf("\nccf_file is opened");
/* locate procedure section in the opened process-plan file */
e="PROCEDURE_SECTION";
if ((i=locate_string(ppfp,e))<0){
    printf("\nprocedure section is not found, back to main menu");
    fclose(ppfp);
    fclose(ccfp);
    quit_auto_flag=ON;
    break;
}
printf("\nprocedure section is found");
/* command recognition and decomposition */
e=">>";
do{
    if ((i=locate_string(ppfp,e))>0){
        /* read in command */
        fscanf(ppfp,"%s",temp);
        j=strlen(temp);
        printf("\ncommand length = %-d",j);
        printf("\ncommand found :%s:",temp);

        /* command recognition */
        str="LOAD_PART";
```

```
                if (strncmp(temp,str,j)==0) {
                    load_part(ppfp,ccfp);
                }
                else {
                    str="UNLOAD_PART";
                    if (strncmp(temp,str,j)==0) {
                        unload_part(ppfp,ccfp);
                    }
                    else {
                        str="MACHINE_PART";
                        if (strncmp(temp,str,j)==0) {
                            machine_part(ppfp,ccfp);
                        }
                        else
                            printf("\nunrecognized command :%s, %s",temp,
                            "skip-over worning");
                    }
                }
            }
        }
        while (i>0);
        fclose(ppfp);
        fclose(ccfp);

        /* execution commands in ccfile */
        openf1=open(ccfn,READMODE);
        printf("\nopenf1 =%d",openf1);
        while (read(openf1,eecp,65)>0)
        {
            strcpy(ecp,eecp);
            printf("\necp is :%s",ecp);
            printf("\ncommand is :%8c",ecp+18);
            /* seperate nc-file download command from others */
            if (strncmp(ecp+18,dwn,8)==0)
            {
                printf("\nI am in download module");
                strcpy(comout,ecp);
                ecpcom(ecp);
                printf("\nreceived ecp:%s",ecp);
                if (strncmp(ecp+12,rdy,6)!=0)
                {
                    printf("\n%s","download procedure err, back to main menu");
                    quit_auto_flag=ON;
                    break;
                }
                strncpy(ncfile,comout+33,15);
                for (i=0;i<15;++i){
                    if (*(ncfile+i) == ' ') {
                        *(ncfile+i) = '\0';
                        *(ncfile+15) = ' ';
                        break;
                    }
                }
                path="/usr/unger/vws2/design/nc/";
```

```
                    strcat(path,ncfile);
                    strcpy(ncfile,path);
                    printf("executing nc_file ID :%s.",ncfile);
                    download(ncfile);
                }
                else
                {
                    strcpy(comout,ecp);
                    ecpcom(ecp);

                    if (strncmp(ecp+12,rdy,6)!=0)
                    {
                        printf("\n%s","ecp status error, back to main menu");
                        quit_auto_flag=ON;
                        break;
                    }
                }
            }
            close(openf1);
            /* scpcomout - response the result of execution */
            if (quit_auto_flag==OFF) printf("\npp_file is successfully done");
        }
        while (quit_auto_flag==OFF);
        break;
    case 3 :       /* case 3 is ccp help mode */
        break;
    case 4 :       /* quit ccp */
        quit_ccp_flag=ON;
    }
}
while (quit_ccp_flag == OFF);
}
```

```
fmlab3% cat stareq.c


/* -----------------------------------------------------------------------*
 * This function is used to perform the ccp/manual/status_request mode of  *
 * ccp program.                                                            *
 * --------------------------------------------------------------------- */


#include <stdio.h>
#define     ON      1
#define     OFF     0


char *ecp,*scp,*comin,*comout,*stabuf[6];


stareq()
{
    char temp[30],*str;
    int atoi(),quit_sta_flag,stamode;


    quit_sta_flag = OFF;
    do{
        /* print ccp/manual/status_request menu and select entry mode */
        printf("\n\n%s%s%s%s%s%s","1. ECP   ",
                                  "2. VMC   ",
                                  "3. VISE   ",
                                  "4. GRIPPER   ",
                                  "5. ROBOT   ",
                                  "6. quit manual/sta_req");
        do{
            printf("\n%s","ccp/manual/sta_req=>");
            scanf("%s",temp);
            stamode=atoi(temp);
            if ((stamode <1) || (stamode > 6)) {
                printf("%s","input error. valid number 1 .. 6");
            }
            else
                break;
        }
        while (1);

        /* branch ccp/manual mode based on stamode */
        switch (stamode){
        case 1 :
            /* request for ECP status */
            str="ECP     ";
            staio(str,stamode);
            break;
        case 2 :
            /* request for VMC status */
```

```
                str="VMC     ";
                staio(str,stamode);
                break;
            case 3 :
                /* request for VISE status */
                str="VISE    ";
                staio(str,stamode);
                break;
            case 4 :
                /* request for GRIPPER status */
                str="GRIPPER ";
                staio(str,stamode);
                break;
            case 5 :
                /* request for ROBOT status */
                str="ROBOT   ";
                staio(str,stamode);
                break;
            case 6 :
                /* quit ccp/manual/sta_req mode */
                quit_sta_flag = ON;
            }
            printf("\necp in stareq.c :%s",ecp);
        }
        while (quit_sta_flag == OFF);
    }
```

```
fmlab4% cat srvreq.c


/* ------------------------------------------------------------------------- *
 * this function is used to perform the ccp/manual/service_request mode of   *
 * ccp program. It communicates with the main program through the parameters *
 * declared at the outmost scope.                                             *
 *                                                                            *
 * Inputs of this function are target selections and target positions       *
 * provided by user. Outputs are the status of selected targets that were    *
 * commanded.                                                                 *
 * --------------------------------------------------------------------------*/


#include <stdio.h>
#define     ON      1
#define     OFF     0


char *ecp,*scp,*comin,*comout,*stabuf[6];


srvreq()
{
    char temp[3][30],*str,*tp[3],tmp[30];
    int i,j,k,atoi(),quit_srv_flag,srvmode;


    /* initialization */
    quit_srv_flag = OFF;
    for (i=0;i<3;++i) tp[i] = "               ";

    /* loop the service_request menu and select the entry mode */
    do{
        /* print ccp/manual/service_request menu */
        printf("\n\n%s%s%s%s%s",
                                "1. VMC   ",
                                "2. VISE   ",
                                "3. GRIPPER   ",
                                "4. ROBOT   ",
                                "5. quit manual/srv_req");
        do{
            printf("\n%s","ccp/manual/srv_req=>");
            scanf("%s",tmp);
            srvmode=atoi(tmp);
            if ((srvmode <1) || (srvmode > 5)) {
                printf("%s","input error. valid number 1 .. 5");
            }
            else
                break;
        }
        while (1);
```

```
/* branch ccp/manual mode dased on srvmode */
switch (srvmode){
case 1 :
    /* request VMC service */
    /* initialize str to a 64-character string, VERY IMPORTANT */
    str = "COM VMC                                                      %";

    /* enter target position coordinates x, y, and z */
    do{
        printf("\nPlease enter VMC target coordinates:");
        printf("\n    x = ");
        scanf("%s",temp[0]);
        printf("\n    y = ");
        scanf("%s",temp[1]);
        printf("\n    z = ");
        scanf("%s",temp[2]);
        printf("\nAre you SURE ? (y/n)");
        scanf("%s",tmp);
    }
    while (tmp[0] != 'y');

    /* format temp to 15 characters a line */
    for (j=0;j<3;++j){
        i=strlen(temp[j]);
        for (k=i;k<15;++k)  temp[j][k] = ' ';
        temp[j][15] = '\0';
    }

    /* form the command line */
    for (i=0;i<3;++i){
        for (j=0;j<15;++j)
            *(str+18+(15*i)+j) = temp[i][j];

    }

    /* copy command to com_out register */
    strcpy(comout,str);
    printf("\ncomout :%s",comout);

    /* copy command to ecp mailbox */
    strcpy(ecp,str);

    /* command execution */
    ecpcom(ecp);

    /* copy the returned message to appropriate status buffer */
    strcpy(stabuf[2],ecp);
    printf("\nvmc_sta :%s",stabuf[2]);
    break;
case 2 :
    /* request VISE service */
    /* initialize str to a 64-character format, VERY IMPORTANT */
    str = "COM VISE                                                     %";
```

```
        /* enter desired action to be executed on vise */
        do{
            printf("\n(o)pen or (c)lose the vise ?");
            scanf("%s",tmp);
        }
        while ((tmp[0] != 'o') && (tmp[0] != 'c'));

        /* format parameter fields */
        if (tmp[0] == 'o')
            tp[0] = "OPEN           ";
        else
            tp[0] = "CLOSE          ";
        tp[1] = "               ";
        tp[2] = "               ";

        /* form the command line */
        for (i=0;i<15;++i){
            *(str+18+i) = *(tp[0]+i);
            *(str+18+15+i) = *(tp[1]+i);
            *(str+18+30+i) = *(tp[2]+i);
        }

        /* copy command to com_out register */
        strcpy(comout,str);
        printf("\ncomout :%s",comout);

        /* copy command to ecp mailbox */
        strcpy(ecp,str);

        /* command execution */
        ecpcom(ecp);

        /* copy the returned message to appropriate status buffer */
        strcpy(stabuf[3],ecp);
        printf("\nvmc_sta :%s",stabuf[3]);
        break;
    case 3 :
        /* request GRIPPER service */
        /* initialize str to a 64-character format, VERY IMPORTANT */
        str = "COM GRIPPER                                              %";

        /* enter desired action to be executed on gripper */
        do{
            printf("\n(o)pen or (c)lose the gripper ?");
            scanf("%s",tmp);
        }
        while ((tmp[0] != 'o') && (tmp[0] != 'c'));

        /* format parameter fields */
        if (tmp[0] == 'o')
            tp[0] = "OPEN           ";
        else
            tp[0] = "CLOSE          ";
        tp[1] = "               ";
        tp[2] = "               ";

        /* form the command line */
        for (i=0;i<15;++i){
            *(str+18+i) = *(tp[0]+i);
            *(str+18+15+i) = *(tp[1]+i);
            *(str+18+30+i) = *(tp[2]+i);
        }

        /* copy command to com_out register */
        strcpy(comout,str);
        printf("\ncomout :%s",comout);

        /* copy command to ecp mailbox */
        strcpy(ecp,str);

        /* command execution */
        ecpcom(ecp);

        /* copy the returned message to appropriate status buffer */
        strcpy(stabuf[4],ecp);
        printf("\nvmc_sta :%s",stabuf[4]);
        break;
    case 4 :
        /* request ROBOT service */
        /* initialize str to a 64-character string, VERY IMPORTANT */
        str = "COM ROBOT                                              %";

        /* enter target position coordinates x, y, and z */
        do{
            printf("\nPlease enter ROBOT target coordinates:");
            printf("\n    x = ");
            scanf("%s",temp[0]);
            printf("\n    y = ");
            scanf("%s",temp[1]);
            printf("\n    z = ");
            scanf("%s",temp[2]);
            printf("\nAre you SURE ? (y/n)");
            scanf("%s",tmp);
        }
        while (tmp[0] != 'y');

        /* format temp to 15 characters a line */
        for (j=0;j<3;++j){
            i=strlen(temp[j]);
            for (k=i;k<15;++k) temp[j][k] = ' ';
            temp[j][15]='\0';
        }

        /* form the command line */
        for (i=0;i<3;++i){
            for (j=0;j<15;++j){
                *(str+18+(15*i)+j) = temp[i][j];
            }
            }

            /*  copy command to com_out register */
            strcpy(comout,str);
            printf("\ncomout :%s",comout);

            /* copy command to ecp mailbox */
            strcpy(ecp,str);

            /* command execution */
            ecpcom(ecp);

            /* copy the returned message to appropriate status buffer */
            strcpy(stabuf[5],ecp);
            printf("\nvmc_sta :%s",stabuf[5]);
            break;
        case 5 :
            /* quit ccp/manual/srv_req mode */
            quit_srv_flag = ON;
        }
    }
    while (quit_srv_flag == OFF);
}
```

```
fmlab5% cat staio.c


/* ------------------------------------------------------------------------ *
 * This function is used to perform status request procedures. The procedures *
 * are as the following:                                                    *
 * 1). define the blank STA type command.                                   *
 * 2). call stafom() to fill out the target and form the command.           *
 * 3). registers the command to applicable registers.                       *
 * 4). call ecpcom() to output/input the command and status.                *
 * 5). copy the latest status information to appropriate status buffer.     *
 *                                                                          *
 * Couple <string.h> commands are used in this function. Note that when use *
 * the strcpy or strcat for a specific target pointer at the first time,    *
 * the target pointer has to be defined beforehand.                         *
 * It may be defined as any string, even a null string (for strcat). However, *
 * using strcpy or strcat without define the target pointer will cause      *
 * errors.                                                                  *
 * ------------------------------------------------------------------------ */


char *ecp,*scp,*comin,*comout,*stabuf[6];


staio(str,stamode)
char *str;
int stamode;
    /* str    - is a pointer carries the target information whom the
                staio get the status information from.
       buf    - is a status buffer specifically assigned for the target
                described in str.
       ecp    - is a pointer which acts as the ecp mailbox.
       comout - is the outgoing command register.                      */

{
    char *temp;

    /* initialize the blank STA type command */
    temp="STA                                                       %";
    printf("\n%s","I am in staio.c, about to form STA command");

    /* form the command line */
    stafom(temp,str);
    printf("\nI am back from stafom to staio");
    printf("\ntemp is :%s",temp);

    /* copy temp to comout register */
    strcpy(comout,temp);
    printf("\ncomout is :%-s",comout);

    /* copy temp to ecp mailbox */
    strcpy(ecp,temp);
    printf("\necp is :%-s",ecp);
    printf("\n%s","I am about to enter ecpcom.c");

    /* communicate with ecp (equipment controller) */
    ecpcom(ecp);            /* send out message and receive answer from ecp */

    /* update the status buffer */
    strcpy(stabuf[stamode],ecp);
    printf("\nstatus_buffer :%s",stabuf[stamode]);
}
```

```
fmlab6% cat stafom.c


/* ---------------------------------------------------------------------- *
 * This function is used to form the status-request-command in the right   *
 * format.                                                                  *
 * ---------------------------------------------------------------------- */


stafom(temp,str)
char *temp,*str;
    /* temp - a pointer of a blank command of specific type
       str  - command target, will be filled into the target field of temp */
{
    int i;

    /* fill the target into the target field of temp */
    printf("\n%s","I am in stafom.c");
    for (i=0;i<8;++i){
        *(temp+4+i) = *(str+i);
    }
    printf("\ntemp is :%s",temp);
}
```

```
fmlab8% cat ecpcom.c


/* ------------------------------------------------------------ *
 * this program serves as the communication module for the ccp *
 * to communicate with ecp of the vertival machining cell      *
 * ------------------------------------------------------------ */



#include <stdio.h>
#include <sys/file.h>
#define READMODE 0444
#define  WRITEMODE 0222

ecpcom(chars)
char *chars;

{
    char  *file1,temp[80];
    int   openf1,i;

    printf("\nI am in ecpcom");
    printf("\n%s",chars);
    file1="/dev/ttyb";
    openf1=open(file1,WRITEMODE);
    for (i=0;i<64;++i){
        temp[i] = *(chars+i);
    }
    temp[64]='\n';
    temp[65]='\0';
    printf("\n%s",temp);
    sleep (1);
    write(openf1,temp,65);
    i=close(openf1);
    if (i==0) printf("\nttyb is closed");

    file1="/dev/ttyb";
    openf1=open(file1,READMODE);
    printf("\nopenf1 =%-d",openf1);
    if (openf1>0) printf("\nopenf1 is opened");
    i=read(openf1,chars,64);
    printf("\n%d chars are successfully converted",i);
    printf("\nchars=%s",chars);
    i=close(openf1);
    if (i==0) printf("\nttyb is closed from reading");
}
```

```
fmlab9% cat download.c


#include <stdio.h>
#define  READMODE 0444
#define  WRITEMODE 0222
#define  ON        1
#define  OFF       0


download(ncfile)
char *ncfile;
{
    int i,openf2,count,quit;
    char temp[80],*f2,*chars;
    FILE *fp,*fopen();
    /* ncfile - name of the file that contains the nc code.
       count - count the length of each line read in from nc file.
       quit - flag for quiting out from reading nc file.
       temp - temporary buffer to store the latest input from nc file.
       chars - temporary buffer to store the latest input from ecp.
       f2 - file name of port b of sun workstation.
       openf2 - file descripter for port b of sun workststion.
       fp - pointer points to the nc file.                              */


    /* initialize the program, delay the output for ecp to creat space */
    quit=OFF;
    count=0;
    chars="                                                                   ";
    sleep (5);

    /* open nc file and port b */
    if ((fp=fopen(ncfile,"r"))!=NULL) {
        printf("\nfile is opened\n");
        f2="/dev/ttyb";
        openf2=open(f2,WRITEMODE);

        /* read in a line of code from nc file and put it to port b */
        while (quit==OFF)
        {
            /* read in a line of code */
            do {
                if (fscanf(fp,"%c",&temp[count])!=1) {
                    quit=ON;
                    printf("\nbreak the download program");
                    break;
                }
                ++count;
            }
            while (temp[count-1]!='\n');

            /* when not EOF, print to port b and to screen */
            if (quit==OFF) {

                /* add end_of_string character to initilize the unused
                   characters. Very Important ! */
                temp[count]='\0';

                write(openf2,temp,count);
                printf("%s",temp);

                /* delay output to port b for synchronization */
                for (i=0;i<30000;++i) {}

                /* recognize the last line in nc file and set flag to ON */
                if (count >0){
                    if ((temp[0]=='%') && (temp[1]<'0')) {
                    quit=ON;
                    printf("\nend of the ncfile");
                    break;
                    }

                }
                count=0;

            }
            else {
                break;
            }
        }
        /* close all opened files and return positive number to calling
           program */
        fclose(fp);
        close(openf2);

        /* wait for the echo from the equipment controller for the completion
           of downloading */
        f2="/dev/ttyb";
        openf2=open(f2,READMODE);
        printf("\nopenf2 =%-d",openf2);
        if (openf2>0) printf("\nopenf2 is opened");
        i=read(openf2,chars,64);
        printf("\n%d chars are successfully converted",i);
        printf("\nchars=%s",chars);
        i=close(openf2);
        if (i==0) printf("\nttyb is closed from reading");
        return(1);
    }
    else {
        /* return negative number to calling program */
        return(-1);
    }
}
```

```
fmlab10% cat locate_string.c


/* ------------------------------------------------------------------ *
 * This program is a function used to locate a string in a pointed file.    *
 * The string and file pointer will be passed from the calling program by   *
 * using "call by referance" in passing arguments.                          *
 *                                                                          *
 *                                                                          *
 * The function will return the number of tries before it reaches the string *
 *  If it fails in finding the string before EOF, a negative value will be   *
 *  returned.                                                               *
 * ------------------------------------------------------------------ */



#include <stdio.h>
/* stdio.h has to be included for cc -c compilation due to the fact that FILE
   declation is checked when the funtion is compiled */

locate_string(fp,str)
/* Notice that ";" is not allowed here */

FILE *fp;
char *str;
{
    int i,flag;
    char a[30];

    i=0;
    /* Counter that counts # of tries */

    flag=0-1;
    /* set negative # to flag (when assign a negative value to an int variable
       , one has to use "expression" rather than assign the value directly */

    while (fscanf(fp,"%s",a)==1)
    {
        ++i;
        if (strcmp(a,str)==0)
        {
            flag=i;
            break;
        }
    }
    return(flag);
}
```

```
fmlab11% cat load_part.c


/* ----------------------------------------------------------------- *
 * This program is a function used to decompose the PP_CCP work element     *
 * LOAD_PART into a set of cell level commands.                              *
 *                                                                            *
 * Argument passed from the main program are pointer of the cell level      *
 * process plan file and pointer of the cell command file (CCF) where it    *
 * reads and writes the data.                                                *
 * Initialized value is the selected setup # of vmc_setup database.          *
 * database. Input is vmc_setup file and output are cell commands printed   *
 * to CCF.                                                                   *
 * ----------------------------------------------------------------- */



#include <stdio.h>

load_part(ppfp,ccfp)
FILE *ppfp,*ccfp;
{
    /* ------------------------ DECLARATION -------------------------- */
    int setup_no,atoi(),i,locate_string();
    /* setuo_no - is a pointer pointing to the current configuration in the
                  vmc_setup file.
       atoi() - is a function that converts ASCII-string into integer */

    float xa,ya,za,xb,yb,zb,xd,yd,zd,xp,yp,zp,vl,vh,w,atof();
    /* xa,ya,za - robot home position, global coordinate system, inches.
       xb,yb,zb - pick-up point (refer to center of gripper tip), global
                  coordinate system, inches.
       xd,yd,zd - target point, global coordinate system, inches.
       xp,yp,zp - vise position. Clamping direction is supposed to be
                  parallel to y axis of machine table. Point is refered to
                  the inner_left tip of vise. Global coordinate system ,
                  inches.
       vl,vh - vise length and vise height. Height is measured between the
               vise base to the vise top.
       w - width of the part.
       atof() - is a function that converts ASCII-string into real number  */

    FILE *fp,*fopen();
    char *a,*e,temp[30],*field[6];


    /* ------------------------ PROGRAM  BODY ------------------------- */
    setup_no=1;
    xa=0.0;      ya=80.0;      za=8.0;
    xb=0.0;      yb=160.0;     zb=2.50;

    /* find the part width from the pointed process_plan file */
    e="PART_SIZE_W";
```

```
locate_string(ppfp,e);
e="=>";
locate_string(ppfp,e);
fscanf(ppfp,"%s",temp);
w=atof(temp);

/* locate the line in vmc_setup file pointed by variable setup_no */
fp=fopen("/usr/chen/automation/vmc_setup","r");
do{
    e="(";
    locate_string(fp,e);
    fscanf(fp,"%s",temp);
  } while ((i=atoi(temp))!=setup_no);

/* read in five numbers from vmc_setup for calculation */
fscanf(fp,"%s",temp);
fscanf(fp,"%s",temp);      xp=atof(temp);
fscanf(fp,"%s",temp);      yp=atof(temp);
fscanf(fp,"%s",temp);      zp=atof(temp);
fscanf(fp,"%s",temp);      vl=atof(temp);
fscanf(fp,"%s",temp);      vh=atof(temp);
fclose("/usr/chen/automation/vmc_setup");

/* calculate the target point */
xd=xp-vl/2;
yd=yp+w/2+0.1;
zd=zp+vh+0.2;
/* printf("\nw=%-f  xp=%-f  yp=%-f  vl=%-f vh=%-f\n\n",w,xp,yp,vl,vh); */

/* cell level command formation */
e="%";

/* move table to machine home position */
a="";
field[0]="COM ";
field[1]="VMC     ";
field[2]="      ";
field[3]="INITIAL        ";
field[4]="               ";
field[5]="               ";
for (i=0;i<=5;++i)
    strcat(a,field[i]);
strcat(a,e);
fprintf(ccfp,"%s\n",a);

/* open the gripper */
a="";
field[0]="COM ";
field[1]="GRIPPER ";
field[2]="      ";
field[3]="OPEN           ";
field[4]="               ";
field[5]="               ";
for (i=0;i<=5;++i)
```

```
    strcat(a,field[i]);
strcat(a,e);
fprintf(ccfp,"%s\n",a);

/* open the vise */
a="";
field[0]="COM ";
field[1]="VISE    ";
field[2]="      ";
field[3]="OPEN            ";
field[4]="                ";
field[5]="                ";
for (i=0;i<=5;++i)
    strcat(a,field[i]);
strcat(a,e);
fprintf(ccfp,"%s\n",a);

/* move robot to 12 inches above pick-up point */
a="";
field[0]="COM ";
field[1]="ROBOT   ";
field[2]="      ";
for (i=0;i<=2;++i)
    strcat(a,field[i]);
fprintf(ccfp,"%s",a);
fprintf(ccfp,"%-15.4f%-15.4f%-15.4f",xb,yb,zb+12);
fprintf(ccfp,"%%\n");

/* move robot to pick-up point */
a="";
field[0]="COM ";
field[1]="ROBOT   ";
field[2]="      ";
for (i=0;i<=2;++i)
    strcat(a,field[i]);
fprintf(ccfp,"%s",a);
fprintf(ccfp,"%-15.4f%-15.4f%-15.4f",xb,yb,zb);
fprintf(ccfp,"%%\n");

/* close gripper to hold the part */
a="";
field[0]="COM ";
field[1]="GRIPPER ";
field[2]="      ";
field[3]="CLOSE           ";
field[4]="                ";
field[5]="                ";
for (i=0;i<=5;++i)
    strcat(a,field[i]);
strcat(a,e);
fprintf(ccfp,"%s\n",a);

/* move the robot to 2 inches above the target point */
a="";
```

```
field[0]="COM ";
field[1]="ROBOT   ";
field[2]="      ";
for (i=0;i<=2;++i)
    strcat(a,field[i]);
fprintf(ccfp,"%s",a);
fprintf(ccfp,"%-15.4f%-15.4f%-15.4f",xd,yd,zd+2);
fprintf(ccfp,"%%\n");

/* move the robot to target point */
a="";
field[0]="COM ";
field[1]="ROBOT   ";
field[2]="      ";
for (i=0;i<=2;++i)
    strcat(a,field[i]);
fprintf(ccfp,"%s",a);
fprintf(ccfp,"%-15.4f%-15.4f%-15.4f",xd,yd,zd);
fprintf(ccfp,"%%\n");


/* release the part */
a="";
field[0]="COM ";
field[1]="GRIPPER ";
field[2]="      ";
field[3]="OPEN           ";
field[4]="               ";
field[5]="               ";
for (i=0;i<=5;++i)
    strcat(a,field[i]);
strcat(a,e);
fprintf(ccfp,"%s\n",a);

/* close the vise to hold the part on the table */
a="";
field[0]="COM ";
field[1]="VISE    ";
field[2]="      ";
field[3]="CLOSE          ";
field[4]="               ";
field[5]="               ";
for (i=0;i<=5;++i)
    strcat(a,field[i]);
strcat(a,e);
fprintf(ccfp,"%s\n",a);

/* move the robot to 2 inches above the target point */
a="";
field[0]="COM ";
field[1]="ROBOT   ";
field[2]="      ";
for (i=0;i<=2;++i)
    strcat(a,field[i]);
    fprintf(ccfp,"%s",a);
    fprintf(ccfp,"%-15.4f%-15.4f%-15.4f",xd,yd,zd+2);
    fprintf(ccfp,"%%\n");

    /* move the robot outside the machining zone */
    a="";
    field[0]="COM ";
    field[1]="ROBOT   ";
    field[2]="      ";
    for (i=0;i<=2;++i)
        strcat(a,field[i]);
    fprintf(ccfp,"%s",a);
    fprintf(ccfp,"%-15.4f%-15.4f%-15.4f",xa,ya,za);
    fprintf(ccfp,"%%\n");
}
```

```
fmlab12% cat machine_part.c


/* -------------------------------------------------------------------- *
 * This program is a function used to decompose the PP_CCP work element   *
 * MACHINE_PART into a set of cell level commands.                        *
 * Argument passed from the main program are pointer of the cell level    *
 * process plan file and pointer of the cell command file (CCF) where it  *
 * reads and writes the data.                                             *
 * -------------------------------------------------------------------- */




#include <stdio.h>

machine_part(ppfp,ccfp)
FILE *ppfp,*ccfp;
{
    int i,locate_string();
    char a[15],*e,*field[6];

    e="NCFILE_NAME";
    locate_string(ppfp,e);
    e="=>";
    locate_string(ppfp,e);
    fscanf(ppfp,"%s",a);

    /* cell level command formation */
    e="%";
    field[0]="COM ";
    field[1]="VMC     ";
    field[2]="      ";
    field[4]="               ";
    field[5]="               ";

    /* download the ncfile to vmc */
    field[3]="DOWNLOAD       ";
    for (i=0;i<=3;++i)
        fprintf(ccfp,"%s",field[i]);
    fprintf(ccfp,"%-15s",a);
    for (i=5;i<=5;++i)
        fprintf(ccfp,"%s",field[i]);
    fprintf(ccfp,"%s\n",e);

    /* preset the vmc */
    field[3]="PRE_SET        ";
    for (i=0;i<=5;++i)
        fprintf(ccfp,"%s",field[i]);
    fprintf(ccfp,"%s\n",e);

    /* command the vmc to start machining */
    field[3]="EXECUTE        ";
    for (i=0;i<=5;++i)
```

```
            fprintf(ccfp,"%s",field[i]);
        fprintf(ccfp,"%s\n",e);
}
```

```
fmlab13% cat unload_part.c


/* ----------------------------------------------------------------- *
 * This program is a function used to decompose the PP_CCP work element    *
 * UNLOAD_PART into a set of cell level commands.                           *
 *                                                                          *
 * Argument passed from the main program are pointer of the cell level     *
 * process plan file and pointer of the cell command file (CCF) where it   *
 * reads and writes the data.                                               *
 * Initialized value is the selected setup # of vmc_setup database.        *
 * database. Input is vmc_setup file and output are cell commands printed  *
 * to CCF.                                                                  *
 * ----------------------------------------------------------------- */




#include <stdio.h>

unload_part(ppfp,ccfp)
FILE *ppfp,*ccfp;
{
    /* ------------------------- DECLARATION --------------------------- */
    int setup_no,atoi(),i,locate_string();
    /* setuo_no - is a pointer pointing to the current configuration in the
                  vmc_setup file.
       atoi() - is a function that converts ASCII-string into integer */

    float xa,ya,za,xb,yb,zb,xd,yd,zd,xp,yp,zp,vl,vh,w,atof();
    /* xa,ya,za - robot home position, global coordinate system, inches.
       xb,yb,zb - pick-up point (refer to center of gripper tip), global
                  coordinate system, inches.
       xd,yd,zd - target point, global coordinate system, inches.
       xp,yp,zp - vise position. Clamping direction is supposed to be
                  parallel to y axis of machine table. Point is refered to
                  the inner_left tip of vise. Global coordinate system ,
                  inches.
       vl,vh - vise length and vise height. Height is measured between the
               vise base to the vise top.
       w - width of the part.
       atof() - is a function that converts ASCII-string into real number  */

    FILE *fp,*fopen();
    char *a,*e,temp[30],*field[6];


    /* ------------------------ PROGRAM  BODY ------------------------- */
    setup_no=1;
    xa=0.0;     ya=80.0;     za=8.0;
    xb=0.0;     yb=160.0;    zb=2.50;

    /* find the part width from the pointed process_plan file */
    e="PART_SIZE_W";
```

```
locate_string(ppfp,e);
e="=>";
locate_string(ppfp,e);
fscanf(ppfp,"%s",temp);
w=atof(temp);

/* locate the line in vmc_setup file pointed by variable setup_no */
fp=fopen("/usr/chen/automation/vmc_setup","r");
do{
    e="(";
    locate_string(fp,e);
    fscanf(fp,"%s",temp);
  } while ((i=atoi(temp))!=setup_no);

/* read in five numbers from vmc_setup for calculation */
fscanf(fp,"%s",temp);
fscanf(fp,"%s",temp);      xp=atof(temp);
fscanf(fp,"%s",temp);      yp=atof(temp);
fscanf(fp,"%s",temp);      zp=atof(temp);
fscanf(fp,"%s",temp);      vl=atof(temp);
fscanf(fp,"%s",temp);      vh=atof(temp);
fclose("/usr/chen/automation/vmc_setup");

/* calculate the target point */
xd=xp-vl/2;
yd=yp+w/2+0.1;
zd=zp+vh+0.2;
/* printf("\nw=%-f  xp=%-f  yp=%-f  vl=%-f vh=%-f\n\n",w,xp,yp,vl,vh); */

/* cell level command formation */
e="%";

/* move table to machine home position */
a="";
field[0]="COM ";
field[1]="VMC     ";
field[2]="      ";
field[3]="INITIAL        ";
field[4]="               ";
field[5]="               ";
for (i=0;i<=5;++i)
    strcat(a,field[i]);
strcat(a,e);
fprintf(ccfp,"%s\n",a);

/* open the gripper */
a="";
field[0]="COM ";
field[1]="GRIPPER ";
field[2]="      ";
field[3]="OPEN           ";
field[4]="               ";
field[5]="               ";
for (i=0;i<=5;++i)
```

```
    strcat(a,field[i]);
strcat(a,e);
fprintf(ccfp,"%s\n",a);

/* move the robot to 2 inches above the target point */
a="";
field[0]="COM ";
field[1]="ROBOT   ";
field[2]="      ";
for (i=0;i<=2;++i)
    strcat(a,field[i]);
fprintf(ccfp,"%s",a);
fprintf(ccfp,"%-15.4f%-15.4f%-15.4f",xd,yd,zd+2);
fprintf(ccfp,"%%\n");

/* move the robot to target point */
a="";
field[0]="COM ";
field[1]="ROBOT   ";
field[2]="      ";
for (i=0;i<=2;++i)
    strcat(a,field[i]);
fprintf(ccfp,"%s",a);
fprintf(ccfp,"%-15.4f%-15.4f%-15.4f",xd,yd,zd);
fprintf(ccfp,"%%\n");

/* close gripper to hold the part */
a="";
field[0]="COM ";
field[1]="GRIPPER ";
field[2]="      ";
field[3]="CLOSE          ";
field[4]="               ";
field[5]="               ";
for (i=0;i<=5;++i)
    strcat(a,field[i]);
strcat(a,e);
fprintf(ccfp,"%s\n",a);


/* open the vise */
a="";
field[0]="COM ";
field[1]="VISE    ";
field[2]="      ";
field[3]="OPEN           ";
field[4]="               ";
field[5]="               ";
for (i=0;i<=5;++i)
    strcat(a,field[i]);
strcat(a,e);
fprintf(ccfp,"%s\n",a);

/* move the robot to 2 inches above the target point */
```

```
a="";
field[0]="COM ";
field[1]="ROBOT   ";
field[2]="      ";
for (i=0;i<=2;++i)
    strcat(a,field[i]);
fprintf(ccfp,"%s",a);
fprintf(ccfp,"%-15.4f%-15.4f%-15.4f",xd,yd,zd+2);
fprintf(ccfp,"%%\n");

/* move robot to 12 inches above pick-up point */
a="";
field[0]="COM ";
field[1]="ROBOT   ";
field[2]="      ";
for (i=0;i<=2;++i)
    strcat(a,field[i]);
fprintf(ccfp,"%s",a);
fprintf(ccfp,"%-15.4f%-15.4f%-15.4f",xb,yb,zb+12);
fprintf(ccfp,"%%\n");

/* move robot to 0.1 inch above pick-up point */
a="";
field[0]="COM ";
field[1]="ROBOT   ";
field[2]="      ";
for (i=0;i<=2;++i)
    strcat(a,field[i]);
fprintf(ccfp,"%s",a);
fprintf(ccfp,"%-15.4f%-15.4f%-15.4f",xb,yb,zb+0.1);
fprintf(ccfp,"%%\n");

/* open the gripper to release the part */
a="";
field[0]="COM ";
field[1]="GRIPPER ";
field[2]="      ";
field[3]="OPEN           ";
field[4]="               ";
field[5]="               ";
for (i=0;i<=5;++i)
    strcat(a,field[i]);
strcat(a,e);
fprintf(ccfp,"%s\n",a);

/* move robot to 12 inches above pick-up point */
a="";
field[0]="COM ";
field[1]="ROBOT   ";
field[2]="      ";
for (i=0;i<=2;++i)
    strcat(a,field[i]);
fprintf(ccfp,"%s",a);
fprintf(ccfp,"%-15.4f%-15.4f%-15.4f",xb,yb,zb+12);
```

```
    fprintf(ccfp,"%%\n");

    /* move the robot to the robot home position */
    a="";
    field[0]="COM ";
    field[1]="ROBOT   ";
    field[2]="      ";
    for (i=0;i<=2;++i)
        strcat(a,field[i]);
    fprintf(ccfp,"%s",a);
    fprintf(ccfp,"%-15.4f%-15.4f%-15.4f",xa,ya,za);
    fprintf(ccfp,"%%\n");
}
```

```
fmlab14% cat scpmail


ready
```

```
fmlab15% cat tempfile


COM ECP            pp_ccp_test                                          %
```

```
10    !    *************************************************************
11    !    *                                                           *
12    !    *                  Equipment Control Program                *
13    !    *                              of                           *
14    !    *              The Vertical Machining Cell                  *
15    !    *                                                           *
16    !    *           Flexible Manufacturing Laboratory               *
17    !    *           Mechanical Engineering Department               *
18    !    *                University of Maryland                     *
19    !    *                College Park, MD 20742                     *
20    !    *                                                           *
21    !    *                          1987                             *
22    !    *                                                           *
23    !    *************************************************************
40    !
50    !
60    !  Main Program ------ User Interface
70    !
80    COM /Comu/ Bufr$(1:4)[64]
85    COM /Stat/ Statu$(1:4)[64]
86    COM /Sun/ Sunbox$[64]
90    DIM A$[15],B$[15],C$[15],Comreg$[64],Ecpsta$[64],D$[170]
100   CONTROL 17,3;9600     ! I/O TO VISE/GRIPPER
105   CONTROL 17,4;26
110   CONTROL 18,3;9600     ! I/O TO ROBOT AND VMC
115   CONTROL 18,4;26
120   CONTROL 19,3;9600     ! I/O TO SUN
125   CONTROL 19,4;26
130   CONTROL 9,3;4800      ! DATA BUS TO VMC
135   CONTROL 9,4;26
149   CLEAR SCREEN
150   PRINT "*****************************************************************
*************"
151   PRINT "*
        *"
152   PRINT "*                      Welcome To The Equipment Control Program
        *"
153   PRINT "*
        *"
154   PRINT "*****************************************************************
*************"
160   PRINT TABXY(8,10),"1) MANUAL MODE"
170   PRINT TABXY(8,12),"2) AUTO MODE"
180   PRINT TABXY(8,14),"3) HELP"
190   PRINT TABXY(8,16),"4) EXIT"
200   INPUT "Please enter the selected number :",Uimode$
210   SELECT Uimode$
220       CASE ="1"
230           GOSUB Manual
231           GOTO 149
240       CASE ="2"
250           GOSUB Auto
251           GOTO 149
```

```
10    !    ***********************************************************
11    !    *                                                         *
12    !    *                 Equipment Control Program               *
13    !    *                            of                           *
14    !    *                The Vertical Machining Cell              *
15    !    *                                                         *
16    !    *            Flexible Manufacturing Laboratory            *
17    !    *            Mechanical Engineering Department            *
18    !    *                 University of Maryland                  *
19    !    *                 College Park, MD 20742                  *
20    !    *                                                         *
21    !    *                           1987                          *
22    !    *                                                         *
23    !    ***********************************************************
40    !
50    !
60    !  Main Program ------ User Interface
70    !
80    COM /Comu/ Bufr$(1:4)[64]
85    COM /Stat/ Statu$(1:4)[64]
86    COM /Sun/ Sunbox$[64]
90    DIM A$[15],B$[15],C$[15],Comreg$[64],Ecpsta$[64],D$[170]
100   CONTROL 17,3;9600    ! I/O TO VISE/GRIPPER
105   CONTROL 17,4;26
110   CONTROL 18,3;9600    ! I/O TO ROBOT AND VMC
115   CONTROL 18,4;26
120   CONTROL 19,3;9600    ! I/O TO SUN
125   CONTROL 19,4;26
130   CONTROL 9,3;4800     ! DATA BUS TO VMC
135   CONTROL 9,4;26
149   CLEAR SCREEN
150   PRINT "*****************************************************************
*************"
151   PRINT "*
       *"
152   PRINT "*                    Welcome To The Equipment Control Program
       *"
153   PRINT "*
       *"
154   PRINT "*****************************************************************
*************"
160   PRINT TABXY(8,10),"1) MANUAL MODE"
170   PRINT TABXY(8,12),"2) AUTO MODE"
180   PRINT TABXY(8,14),"3) HELP"
190   PRINT TABXY(8,16),"4) EXIT"
200   INPUT "Please enter the selected number :",Uimode$
210   SELECT Uimode$
220       CASE ="1"
230           GOSUB Manual
231           GOTO 149
240       CASE ="2"
250           GOSUB Auto
251           GOTO 149
```

```
260        CASE ="3"
270            GOSUB Help
271            GOTO 149
280        CASE ="4"
290            GOSUB Exit
300        CASE ELSE
305            BEEP
310            PRINT TABXY(1,20),"** Input Error!   Retry after BEEP! **"
320            WAIT 2
330            BEEP
331            PRINT TABXY(1,20),"                                        "
340            GOTO 200
350        END SELECT
355 GOTO 3000
995 ! ------------------------------------------------------------------
999 !
1000 Manual: ! Sub Program of User Interface Main Program.
1010 CLEAR SCREEN
1050 CALL Statuss
1060 CALL Status_print
1070 FOR Tar=1 TO 4
1080     GOSUB Compocheck
1090 NEXT Tar
1150 PRINT TABXY(1,10);"COMMAND TARGET :"
1160 PRINT TABXY(5,11);"1). VMC      2). VISE      3). GRIPPER    4). ROBOT"
1165 REPEAT
1170     INPUT "Please enter the selected number :",Tar
1175 UNTIL (Tar<=4 AND Tar>=1)
1176 IF Statu$(Tar)[13;6]<>"READY " THEN
1177     PRINT TABXY(5,18);Statu$(Tar)[5;8];"is not controllable at this momont.
 Please read the status table."
1178     GOTO 1345
1179 ELSE
1180     Statu$(Tar)[1,64]=Statu$(Tar)[1,12]&"BUSY  "&Statu$(Tar)[19,64]
1181     PRINT TABXY(1,Tar+4);Statu$(Tar)[5,63]
1182 END IF
1184 SELECT Tar
1185     CASE =1
1190         CALL Para1(A$,B$,C$)
1200         Comreg$[1,64]="COM VMC            "&A$&B$&C$&"%"
1210         Bufr$(1)[1,64]=Comreg$
1211         PRINT TABXY(1,12);"                         Monitoring VMC Execution
...                        "
1220         CALL Vmccom
1230         GOTO 1330
1240     CASE =2
1245         CALL Para2(A$)
1250         Comreg$[1,64]="COM VISE          "&A$&"
  "&"%"
1255         Bufr$(2)[1,64]=Comreg$
1256         PRINT TABXY(1,12);"                         Monitoring Vise Execution
 ...                       "
1260         CALL Viscom
1265         GOTO 1330
```

```
1270     CASE =3
1275         CALL Para3(A$)
1280         Comreg$[1,64]="COM GRIPPER        "&A$&"
  "&"%"
1285         Bufr$(3)[1,64]=Comreg$
1286         PRINT TABXY(1,12);"                         Monitoring Gripper Execut
ion ...                          "
1290         CALL Grpcom
1295         GOTO 1330
1300     CASE =4
1305         CALL Para4(A$,B$,C$)
1310         Comreg$[1,64]="COM ROBOT          "&A$&B$&C$&"%
1315         Bufr$(4)[1,64]=Comreg$
1316         PRINT TABXY(1,12);"                         Monitoring Robot Executio
n ...                            "
1320         CALL Rbtcom
1330 END SELECT
1335 Statu$(Tar)[1,64]=Bufr$(Tar)[1,64]
1340 PRINT TABXY(1,Tar+4);Statu$(Tar)[5,63]
1342 PRINT TABXY(1,12);"
                   "
1343 GOSUB Compocheck
1345 REPEAT
1350     INPUT "More command ? (Y)es or (N)o",F$
1355 UNTIL (F$="Y" OR F$="N")
1356 PRINT TABXY(1,18);"
                   "
1360 IF F$="Y" THEN GOTO 1150
1490 RETURN
1495 !
1496 !
1499 !
1500 Auto:   ! Sub Program of User Interface Main Program.
1505 CLEAR SCREEN
1510 CALL Statuss
1515 CALL Status_print
1520 GOSUB Ecp_status
1525 PRINT TABXY(1,13);"                         ECP STANDBY
                   "
1526 WAIT .1
1527 RESET 19
1530 ENTER 19;Sunbox$
1540 PRINT TABXY(1,12);"Command Received :"
1550 PRINT TABXY(1,13);Sunbox$[1,64]
1560 PRINT TABXY(1,14);"-------------------------------------------------------
--------"
1570 Comreg$[1,64]=Sunbox$[1,64]
1580 Comtype$=Sunbox$[1,3]
1590 SELECT Comtype$
1600     CASE ="STA"
1610         IF Sunbox$[5,12]="ECP     " THEN
1620             CALL Statuss
1630             CALL Status_print
1640             GOSUB Ecp_status
```

```
1650                Sunbox$[1,64]=Ecpsta$[1,64]
1670            ELSE
1680                GOSUB Comexe
1690            END IF
1700            GOTO 1780
1710        CASE ="COM"
1720            IF Sunbox$[19;8]="DOWNLOAD" THEN
1730                CALL File_relay
1740                Sunbox$[1,64]="STA"&Sunbox$[4,64]
1750            ELSE
1760                GOSUB Comexe
1770            END IF
1780 END SELECT
1790 FOR I=12 TO 14
1800        PRINT TABXY(1,I);"
                              "
1810 NEXT I
1820 OUTPUT 19;Sunbox$
1830 GOTO 1515
1990 RETURN
1995 !
1996 !
1999 !
2410 Help:    ! Sub Program of User Interface Main Program.
2415 CLEAR SCREEN
2420 PRINT TABXY(8,14),"HELP MODE is not installed yet."
2425 PRINT TABXY(8,20),"Push the F2 key to continue ..."
2430 PAUSE
2435 RETURN
2495 !
2496 !
2499 !
2500 Exit:    ! Sub Program of User Interface Main Program.
2510 CLEAR SCREEN
2520 PRINT TABXY(8,14),"Bye! Have a nice day!"
2590 RETURN
2597 !
2598 !
2599 !
2601 Compocheck:       !CHECK COMPONENT STATUS.
2610 WHILE Statu$(Tar)[13;6]<>"READY "
2615        PRINT TABXY(1,10);"
                              "
2620        PRINT TABXY(1,11);"THE ";Statu$(Tar)[5,12]; 'IS ";Statu$(Tar)[13;5];"! (
R)etry or (I)gnore ?                          "
2630        REPEAT
2640            INPUT Cc$
2650        UNTIL (Cc$="R" OR Cc$="I")
2660        IF Cc$="I" THEN GOTO 2831
2665        PRINT TABXY(1,11);"PLEASE CHECK THE ";Statu$(Tar)[5,12];"AND THEN PRESS
 THE F2 KEY.                  "
2666        PRINT TABXY(1,13);"PLEASE MAKE SURE THE ";Statu$(Tar)[5,12];"IS STANDBY
 !"
2670        PAUSE
```

```
2671     PRINT TABXY(1,13);"
                         "
2675     PRINT TABXY(1,11);"                           CHECKING ...
                         "
2680     Bufr$(Tar)[1,64]="STA"&Bufr$(Tar)[4,64]
2690     SELECT Tar
2700         CASE =1
2710             CALL Vmccom
2720             GOTO 2811
2730         CASE =2
2740             CALL Viscom
2750             GOTO 2811
2760         CASE =3
2770             CALL Grpcom
2780             GOTO 2811
2790         CASE =4
2800             CALL Rbtcom
2810     END SELECT
2811     Statu$(Tar)[1,64]=Bufr$(Tar)[1,64]
2813     PRINT TABXY(1,Tar+4);Statu$(Tar)[5,63]
2820     ! Go to the top of this sub. program to re-test the component."
2830 END WHILE
2831 PRINT TABXY(1,11);"
                     "
2840 RETURN
2895 !
2896 !
2897 !
2900 Ecp_status:          ! CHECK ECP STATUS BASE ON THE COMPONENT STATUS.
2905 Ecpsta$[1,64]="STA ECP     READY
  %"
2910 FOR I=1 TO 4
2915     IF Statu$(I)[13;6]<>"READY " THEN Ecpsta$[1,64]="STA ECP     DOWN  "&Ec
psta$[19,64]
2920 NEXT I
2935 RETURN
2945 !
2946 !
2947 !
2950 Comexe:        ! EXECUTE THE "STA" COMMAND FOR COMPONENT STATUS REQUEST AND
                  "COM" COMMAND FOR EXECUTABLE COMMAND.
2960 Tar$=Sunbox$[5,12]
2970 SELECT Tar$
2980     CASE ="VMC     "
2990         IF ((Comtype$="COM") AND (Statu$(1)[13;6]<>"READY ")) THEN
3000             Sunbox$[1,64]=Statu$(1)[1,64]
3010             GOTO 3370
3020         END IF
3030         Bufr$(1)[1,64]=Sunbox$[1,64]
3040         CALL Vmccom
3050         Statu$(1)[1,64]=Bufr$(1)[1,64]
3060         Sunbox$[1,64]=Statu$(1)[1,64]
3070         GOTO 3370
```

```
3080     CASE ="VISE     "
3090         IF ((Comtype$="COM") AND (Statu$(2)[13,18]<>"READY ")) THEN
3100             Sunbox$[1,64]=Statu$(2)[1,64]
3110             GOTO 3370
3120         END IF
3130         Bufr$(2)[1,64]=Sunbox$[1,64]
3140         CALL Viscom
3150         Statu$(2)[1,64]=Bufr$(2)[1,64]
3160         Sunbox$[1,64]=Statu$(2)[1,64]
3170         GOTO 3370
3180     CASE ="GRIPPER "
3190         IF ((Comtype$="COM") AND (Statu$(3)[13;6]<>"READY ")) THEN
3200             Sunbox$[1,64]=Statu$(3)[1,64]
3210             GOTO 3370
3220         END IF
3230         Bufr$(3)[1,64]=Sunbox$[1,64]
3240         CALL Grpcom
3250         Statu$(3)[1,64]=Bufr$(3)[1,64]
3260         Sunbox$[1,64]=Statu$(3)[1,64]
3270         GOTO 3370
3280     CASE ="ROBOT   "
3290         IF ((Comtype$="COM") AND (Statu$(4)[13;6]<>"READY ")) THEN
3300             Sunbox$[1,64]=Statu$(4)[1,64]
3310             GOTO 3370
3320         END IF
3330         Bufr$(4)[1,64]=Sunbox$[1,64]
3340         CALL Rbtcom
3350         Statu$(4)[1,64]=Bufr$(4)[1,64]
3360         Sunbox$[1,64]=Statu$(4)[1,64]
3370 END SELECT
3380 RETURN
3900 END
3901 !-----------------------------------------------------------------
3902 !                End of the ECP main program
3904 !-----------------------------------------------------------------
3905 !
3906 !
3910 SUB Statuss    ! Status checking.
3911 COM /Comu/ Bufr$(1:4)[64]
3912 COM /Stat/ Statu$(1:4)[64]
3913 Bufr$(1)[1,64]="STA VMC
    %"
3914 Bufr$(2)[1,64]="STA VISE
    %"
3915 Bufr$(3)[1,64]="STA GRIPPER
    %"
3916 Bufr$(4)[1,64]="STA ROBOT
    %"
3918 PRINT TABXY(25,18);"Checking the VMC ...."
3919 CALL Vmccom
3920 PRINT TABXY(25,18);"Checking the VISE ...."
3921 CALL Viscom
3923 PRINT TABXY(25,18);"Checking the GRIPPER ...."
```

```
3925 CALL Grpcom
3928 PRINT TABXY(25,18);"Checking the ROBOT ...."
3930 CALL Rbtcom
3935 FOR I=1 TO 4
3938     Statu$(I)[1,64]=Bufr$(I)[1,64]
3940 NEXT I
3942 SUBEND
3947 !
3948 !
3949 !
3955 SUB Status_print   ! Print out the cell status on the screen.
3956 COM /Comu/ Bufr$(1:4)[64]
3957 COM /Stat/ Statu$(1:4)[64]
3960 CLEAR SCREEN
3962 PRINT TABXY(1,1);"CELL COMPONENT STATUS TABLE:"
3964 PRINT TABXY(1,2);"-----------------------------------------------------------
--"
3966 PRINT TABXY(1,3);"NAME    STATUS          PARAMETERS"
3968 PRINT TABXY(1,4);"-----------------------------------------------------------
--"
3970 FOR I=1 TO 4
3972     PRINT TABXY(1,I+4);Statu$(I)[5,63]
3974 NEXT I
3976 PRINT TABXY(1,9);"-----------------------------------------------------------
--"
3978 SUBEND
4000 !
4001 !
4002 !
4004 SUB Vmccom   ! VMC Communication -- Command output, status Input.
4005 COM /Comu/ Bufr$(1:4)[64]
4007 DIM X$[15],Y$[15],Z$[15]
4010 ! ON TIMEOUT 18,60 GOTO 4300
4030 Aa$=Bufr$(1)[1,3]
4040 SELECT Aa$
4050     CASE ="STA"      ! CHECK COMMAND TYPE
4060         OUTPUT 18;Bufr$(1)
4070         ENTER 18;Bufr$(1)[1,64]
4072         IF Bufr$(1)[1,1]<CHR$(65) THEN Bufr$(1)[1,64]=Bufr$(1)[2,64]&"%"
4080         GOTO 4310
4090     CASE ="COM"
4100         DISABLE
4110         OUTPUT 18;Bufr$(1)
4115         ENTER 18;Bufr$(1)[1,64]
4125         IF Bufr$(1)[1,1]<CHR$(65) THEN Bufr$(1)[1,64]=Bufr$(1)[2,64]&"%"
4210         GOTO 4310
4220     CASE ELSE
4230         PRINT "Unrecognizable command type !"
4235         Bufr$(1)[1,64]="UNRECOGNIZABLE COMMAND TYPE"
4240         GOTO 4310
4250 END SELECT
4300 PRINT TABXY(1,12);"                         VMC/Human is not on"
4305 Bufr$(1)[1,64]="STA%VMC     OFF    "&"
     "&"%"
```

```
4310 SUBEND
4500 !
4501 !
4502 !
4510 SUB Viscom     ! VISE Communication -- Command output, status input.
4511 COM /Comu/ Bufr$(1:4)[64]
4520 ! ON TIMEOUT 17,60 GOTO 4640
4530 Aa$=Bufr$(2)[1,3]
4540 SELECT Aa$
4550     CASE ="STA","COM"
4560         OUTPUT 17;Bufr$(2)
4570         ENTER 17 USING "64A";Bufr$(2)[1,64]
4572         IF Bufr$(2)[1,1]<CHR$(65) THEN Bufr$(2)[1,64]=Bufr$(2)[2,64]&"%"
4580         GOTO 4660
4590     CASE ELSE
4600         PRINT "Unrecognizable command type !"
4610         Bufr$(2)[1,64]="UNRECOGNIZABLE COMMAND TYPE"
4620         GOTO 4660
4630 END SELECT
4640 PRINT TABXY(1,12);"                          VISE/Human is not on"
4650 Bufr$(2)[1,64]="STA VISE    OFF    "&"
    "&"%"
4660 SUBEND
5000 !
5001 !
5002 !
5005 SUB Grpcom     ! GRIPPER Communication -- Command output, status input.
5006 COM /Comu/ Bufr$(1:4)[64]
5020 ! ON TIMEOUT 17,60 GOTO 5140
5030 Aa$=Bufr$(3)[1,3]
5040 SELECT Aa$
5050     CASE ="STA","COM"
5060         OUTPUT 17;Bufr$(3)
5070         ENTER 17 USING "64A";Bufr$(3)[1,64]
5072         IF Bufr$(3)[1,1]<CHR$(65) THEN Bufr$(3)[1,64]=Bufr$(3)[2,64]&"%"
5080         GOTO 5160
5090     CASE ELSE
5100         PRINT "Unrecognizable command type !"
5110         Bufr$(3)[1,64]="UNRECOGNIZABLE COMMAND TYPE"
5120         GOTO 5160
5130 END SELECT
5140 PRINT TABXY(1,12);"                          GRIPPER/Human is not on"
5150 Bufr$(3)[1,64]="STA GRIPPER OFF    "&"
    "&"%"
5160 SUBEND
5500 !
5501 !
5502 !
5513 SUB Rbtcom     ! ROBOT Communication -- Command output, status input.
5514 COM /Comu/ Bufr$(1:4)[64]
5520 ! ON TIMEOUT 18,60 GOTO 5640
5530 Aa$=Bufr$(4)[1,3]
5540 SELECT Aa$
5550     CASE ="STA","COM"
```

```
5560          OUTPUT 18;Bufr$(4)
5570          ENTER 18;Bufr$(4)[1,64]
5572          IF Bufr$(4)[1,1]<CHR$(65) THEN Bufr$(4)[1,64]=Bufr$(4)[2,64]&"%"
5580          GOTO 5660
5590      CASE ELSE
5600          PRINT "Unrecognizable command type !"
5610          Bufr$(4)[1,64]="UNRECOGNIZABLE COMMAND TYPE"
5620          GOTO 5660
5630 END SELECT
5640 PRINT TABXY(1,12);"                          ROBOT/Human is not on"
5650 Bufr$(4)[1,64]="STA ROBOT   OFF   "&"
     "&"%"
5660 SUBEND
6000 !
6001 !
6004 SUB Para1(A$,B$,C$)    ! Input parameters for VMC table -- X,Y,Z coord.
6005 REPEAT
6006     INPUT "(C)ontrol or (I)nitialize the VMC ?",A1$
6007 UNTIL (A1$="C" OR A1$="I")
6008 IF A1$="I" THEN
6009     A$[1,15]="INITIAL        "
6010     B$[1,15]=A$[1,15]
6011     C$[1,15]=A$[1,15]
6012     GOTO 6130                  !GOTO 6110
6014 END IF
6015 REPEAT
6020     INPUT "Enter the X-coordinate of table:",A
6030 UNTIL A<=12
6040 REPEAT
6050     INPUT "Enter the Y-coordinate of table:",B
6060 UNTIL B<=12
6070 REPEAT
6080     INPUT "Enter the Z-coordinate of table:",C
6090 UNTIL C<=12
6100 A$[1,15]=VAL$(A)
6110 B$[1,15]=VAL$(B)
6120 C$[1,15]=VAL$(C)
6130 SUBEND
6300 !
6301 !
6302 !
6305 SUB Para2(A$)    ! Input parameters for VISE -- OPEN,CLOSE, INITIAL.
6310 REPEAT
6320     INPUT "(C)lose, (O)pen or (I)nitialize the vise ?",A1$
6330 UNTIL ((A1$="C" OR A1$="O") OR (A1$="I"))
6340 IF A1$="C" THEN
6350     A$[1,15]="CLOSE          "
6360 ELSE
6361     IF A1$="O" THEN
6370         A$[1,15]="OPEN           "
6371     ELSE
6372         A$[1,15]="INITIAL        "
6373     END IF
6380 END IF
```

```
6390 SUBEND
6600 !
6601 !
6602 !
6605 SUB Para3(A$)      ! Input parameters for GRIPPER -- OPEN, CLOSE, INITIAL.
6610 REPEAT
6620      INPUT "(C)lose, (O)pen or (I)nitialize the gripper ?",A1$
6630 UNTIL ((A1$="O" OR A1$="C") OR (A1$="I"))
6640 IF A1$="C" THEN
6650      A$[1,15]="CLOSE          "
6660 ELSE
6661      IF A1$="O" THEN
6670          A$[1,15]="OPEN           "
6671      ELSE
6672          A$[1,15]="INITIAL        "
6673      END IF
6680 END IF
6690 SUBEND
6900 !
6901 !
6902 !
6903 ! Input parameters for ROBOT -- X, Y, Z coordinates.
6905 SUB Para4(A$,B$,C$)     ! Input parameters for ROBOT -- X,Y,Z coord.
6908 REPEAT
6909      INPUT "(C)ontrol or (I)nitialize the robot ?",A1$
6910 UNTIL (A1$="C" OR A1$="I")
6911 IF A1$="I" THEN
6912      A$[1,15]="INITIAL        "
6913      B=0
6914      C=0
6915      GOTO 7010
6916 END IF
6919 REPEAT
6920      INPUT "Enter the X-coordinate :",A
6930 UNTIL A<=50
6940 REPEAT
6950      INPUT "Enter the Y-coordinate :",B
6960 UNTIL B<=48
6970 REPEAT
6980      INPUT "Enter the Z-coordinate :",C
6990 UNTIL C<=48
7000 A$[1,15]=VAL$(A)
7010 B$[1,15]=VAL$(B)
7020 C$[1,15]=VAL$(C)
7030 SUBEND
7115 !
7116 !
7117 !
7200 SUB File_relay        ! RECEIVE THE NC FILE FROM CCP, SAVE IT TO THE DISC,
                             AND DOWNLOAD IT TO THE VMC.
7210 COM /Comu/ Bufr$(1:4)[64]
7220 COM /Sun/ Sunbox$[64]
7230 DIM D$[80]
7240 ASSIGN @Buff TO BUFFER [51200]
```

```
7250 ASSIGN @Source TO 19
7260 ASSIGN @Dest TO 9
7270 PURGE "NCFILE:,1400,1"
7280 CREATE BDAT "NCFILE:,1400,1",200     ! 50K BYTE =(200 REC)*(256 BYTE/REC)
7290 ASSIGN @Ncfile TO "NCFILE:,1400,1"
7291 Sunbox$[1;64]="STA VMC      READY "&Sunbox$[19,64]
7300 OUTPUT @Source;Sunbox$       ! ECHO BACK TO CCP.
7301 WAIT .1
7302 RESET 19
7310 ENTER @Source;D$         ! ASSIGN THE INBOUND NC CODE TO D$.
7320 OUTPUT @Buff;D$          ! WRITE D$ ONTO THE HIGH SPEED MEMORY BUFFER.
7330 IF ((D$[1,1]<>"%") OR (D$[2,2]>CHR$(48))) THEN GOTO 7310
7340 TRANSFER @Buff TO @Ncfile     ! DUMP THE CONTENTS OF BUFFER ONTO THE DISC.
7350 WAIT 6
7360 Bufr$(1)[1,64]="COM VMC      READY DOWNLOAD
    %"                        ! SIGNAL THE VMC TO PREPARE RECEIVING DATA FROM ECP.
7370 CALL Vmccom              ! SEND OUT THE SIGNAL AND RECEIVE ECHO FROM VMC.
7380 CONTROL @Buff,5;1        ! SET THE EMPTY POINTER TO THE FIRST BYTE OF BUFFER.
7390 ENTER @Buff;D$
7391 PRINT D$
7400 OUTPUT @Dest;D$
7500 IF ((D$[1,1]<>"%") OR (D$[2,2]>CHR$(48))) THEN GOTO 7390
7510 ASSIGN @Buff TO *
7520 ASSIGN @Source TO *
7530 ASSIGN @Dest TO *
7540 ASSIGN @Ncfile TO *
7542 CLEAR SCREEN
7550 SUBEND
```

```
10 REM VMC / ROBOT CONTROLLER SIMULATION PROGRAM
20 CLS
30 PRINT
   "********************************************************
   *************"
40 PRINT "*
   *"
50 PRINT "*                       VMC / Robot Controller
   *"
60 PRINT "*
   *"
70 PRINT "*                         Simulation Program
   *"
80 PRINT "*
   *"
90 PRINT
   "********************************************************
   *************"
100 OPEN "com1:9600,e,7,1,cs3000,cd,ds" AS #1
110 OPEN "HISTORY.DOC" FOR OUTPUT AS #2
120 LOCATE 12,1:PRINT "                         VMC / Robot
    Standby "
130 LINE INPUT #1,A$
140 IF LEFT$(A$,1)=CHR$(10) THEN A$=MID$(A$,2)
150 LOCATE 12,1:PRINT "Command Received :
    "
160 PRINT "*";A$;"*";:PRINT
170 PRINT"--------------------------------------------------
    ----------------"
180 PRINT #2,"----------------------------------------------
    -------------------"
190 PRINT #2,DATE$;"           ";TIME$
200 PRINT #2,A$
210 LOCATE 17,1:PRINT "
    "
220 IF LEFT$(A$,3)<>"COM" THEN GOTO 290
221 MID$(A$,1,3)="STA"
222 IF MID$(A$,19,8)="DOWNLOAD" THEN GOSUB 460
224 IF MID$(A$,19,7)="PRE_SET" THEN GOSUB 500
226 IF MID$(A$,19,7)="EXECUTE" THEN GOSUB 520
238 IF MID$(A$,19,7)="INITIAL" THEN GOTO 350
240 LOCATE 17,1:PRINT "Enter (r)eady or (d)own for
    ";MID$(A$,4,6);" status:";
250 INPUT C$
260 IF C$="r" THEN MID$(A$,13,6)="READY ":GOTO 400
270 IF C$="d" THEN MID$(A$,13,6)="DOWN  ":GOTO 400
280     GOTO 240
290 IF LEFT$(A$,3)<>"STA" THEN GOTO 390
310 LOCATE 17,1:INPUT "Please enter the X position :",A1$
315 LOCATE 18,1:INPUT "Please enter the Y position :",A2$
320 LOCATE 19,1:INPUT "Please enter the Z position :",A3$
325 LOCATE 21,1:INPUT "Are youy sure?",A4$ :IF A4$<>"y" THEN
    GOTO 310
```

```
330 LOCATE 17,1:PRINT "
    "
335 LOCATE 18,1:PRINT "
    "
340 LOCATE 19,1:PRINT "
    "
345 LOCATE 21,1:PRINT "
    ":GOTO 370
350 A1$="0              ": A2$="0              ": A3$="0
    "
370 MID$(A$,19,15)=A1$:MID$(A$,34,15)=A2$:MID$(A$,49,15)=A3$
380 GOTO 240
390 LOCATE 17,1:INPUT "I/O Error. Please check the I/O and
    press RETURN key.",C$:GOTO 210
400 LOCATE 14,1:PRINT "
    "
410 LOCATE 13,1:PRINT "
    "
420 LOCATE 17,1:PRINT "
    "
430 PRINT #2,A$
440 PRINT #1,A$;CHR$(10)
450 GOTO 120
460 LOCATE 16,1:PRINT "Please follow the following procedure
    to download data:"
470 LOCATE 17,1:PRINT "1. set the vmc to edit mode."
471 LOCATE 18,1:PRINT "2. turn the edit lock of VMC to OFF.
472 LOCATE 19,1:PRINT "3. press the IN key on VMC control
    panel."
473 LOCATE 21,1:INPUT "press the <CR> when completed ...",F$
474 FOR I=16 TO 21 : LOCATE I,1 : PRINT "
    " : NEXT I
480 RETURN
500 LOCATE 16,1:PRINT "Please follow the following procedure
    to pre-set VMC:"
501 LOCATE 17,1:PRINT "1. turn the handle-driven mode."
502 LOCATE 18,1:PRINT "2. move the tool to the lower-left
    corner."
503 LOCATE 19,1:PRINT "3. rewrite plateform x,y coordinates
    in edit mode."
504 LOCATE 20,1:PRINT "4. offset tool length.
505 LOCATE 22,1:INPUT "press the <CR> when completed ...",F$
506 FOR I=16 TO 22 : LOCATE I,1 : PRINT "
    " : NEXT I
510 RETURN
520 LOCATE 16,1:PRINT "Please follow the following procedure
    to execute the machining:"
521 LOCATE 17,1:PRINT "1. turn to  memory mode."
522 LOCATE 18,1:PRINT "2. push the cycle-start buttom."
525 LOCATE 20,1:INPUT "press the <CR> when completed ...",F$
526 FOR I=16 TO 20 : LOCATE I,1 : PRINT "
    " : NEXT I
550 RETURN
560 END
```

```
10 REM Vise / Gripper Controller simulation program.
11 CLS
12 PRINT
   "********************************************************
   ************"
13 PRINT "*
   *"
14 PRINT "*                    Vise / Gripper Controller
   *"
15 PRINT "*
   *"
16 PRINT "*                        Simulation Program
   *"
17 PRINT "*
   *"
18 PRINT
   "********************************************************
   ************"
20 OPEN "com1:9600,e,7,1,cs3000,cd,ds" AS #1
30 OPEN "HISTORY.DOC" FOR OUTPUT AS #2
35 LOCATE 12,1:PRINT "                      Vise/Gripper
   Standby"
40 LINE INPUT #1,A$   :REM Wait for and receive the ECP
   command.
45 IF LEFT$(A$,1)=CHR$(10) THEN A$=MID$(A$,2):REM Take off
   the leading "LF".
50 LOCATE 12,1:PRINT "Command Received :
   "
60 PRINT "*";A$;"*";:PRINT    :REM Print the received
   command on the screen.
65 PRINT "--------------------------------------------------
   ----------------"
70 REM Write the date,time and received command into
   "HISTORY.DOC" file.
78 PRINT #2,"-----------------------------------------------
   ------------------"
80 PRINT #2,DATE$;"      ";TIME$
100 PRINT #2,A$
105 LOCATE 17,1:PRINT "
    "
109 REM Command type checking.
110 IF (LEFT$(A$,3)="COM") OR (LEFT$(A$,3)="STA") THEN GOTO
    130
120 LOCATE 17,1:INPUT "I/O Error. Please check I/O and press
    RETURN key.",C$
129 REM Human intervention (input "r" or "d" for ready or
    down).
130 LOCATE 17,1:INPUT "Enter (r)eady or (d)own for device
    status:";C$
140 IF C$="r" THEN MID$(A$,13,6)="READY ":GOTO 170
150 IF C$="d" THEN MID$(A$,13,6)="DOWN  ":GOTO 170
160      GOTO 130
169 REM Status checking requested by ECP. Always response
    with "OPEN".
```

```
170 IF LEFT$(A$,3)="STA" THEN MID$(A$,19,5)="OPEN "
174 REM Change the ECP command format into controller
    response format.
175 IF LEFT$(A$,3)="COM" THEN MID$(A$,1,3)="STA"
177 IF MID$(A$,19,7)="INITIAL" THEN MID$(A$,19,7)="OPEN    "
180 LOCATE 14,1:PRINT "
    "
185 LOCATE 13,1:PRINT "
    "
187 LOCATE 17,1:PRINT "
    "
190 PRINT #2,A$   :REM Write the controller response to
    "HISTORY.DOC" file.
200 PRINT #1,A$;CHR$(13);CHR$(10)   :REM Write the
    controller response to ECP.
210 GOTO 35
220 END
```

CURRICULUM VITAE

Name: Sujen Chen

Permanent address: 4325 Rowalt Drive
Apt. #201
College Park, MD 20740

Degree and date to be conferred: M.S., 1988

Date of birth: May 28, 1961

Place of birth: Taipei, Taiwan, R. O. C.

Secondary education: The Second Boys' High School of Tainan, Taiwan, June 1979

College institutions	Dates	Degree	Date of Degree
Feng Chia University	1979-83	B.S.	June 1983
University of Maryland	1985-88	M.S.(M.E.)	May 1988

Major: Mechanical Engineering

Professional positions held: Engineering Assistant, Aeronautical Research Laboratory, Chung-Shan Institute of Science and Technology, Taiwan.