A control methodology for automated manufacturing

Saeid Nahavandi

A Control Methodology for Automated Manufacturing

A thesis presented for the degree of

Doctor of Philosophy

By

SAEID NAHAVANDI

University of Durham
School of Engineering and Applied Science

April 1991

ABSTRACT

The application of computers in the manufacturing industry has substantially altered the control procedures used to program a whole manufacturing process. Currently, one the problems which automated manufacturing systems are experiencing is the lack of a good overall control system. The subject of this research has been centred on the identification of the problems involved in current methods of control and their advantages and disadvantages in an automated manufacturing system. As a result, a different type of control system has been proposed which distributes both the control and the decision making. This control model is an hybrid of hierarchical and heterarchical control systems which takes advantage of the best points offered by both types of control structures.

The Durham FMS rig has been used as a testbed for an automated manufacturing system to which the hybrid control system has been applied. The implementation of this control system would not have been possible without the design and development of a System Integration Tool (SIT). The system is capable of real-time scheduling of the system activities. Activities within the system are monitored in real-time and a recording of the system events is available, which allows the user to analyse the activities of the system off-line. A network independent communication technique was developed for the Durham FMS which allowed the manufacturing cells to exercise peer-to-peer communication. The SIT also allowed the integration of equipment from different vendors in the FMS.

SUMMARY

This thesis presents a control methodology for an automated manufacturing system. The proposed control system is a hybrid of hierarchical and heterarchical control system which utilises the best points of both systems. This control system was implemented on the experimental FMS rig at Durham university. The control system distributes both the decision making and the control.

The hybrid control system was applied to the FMS by means of a System Integration Tool (SIT) which was designed and developed as part of this research. The SIT allows the programmable devices from a wide variety of vendors to be integrated into a single system. The SIT is comprised of several modules, each responsible for a specific type of functionality in the system. It is designed and developed in a modular fashion where the interfacing of each module to another is performed with the minimum amount of effort.

The SIT allows the user to define its own manufacturing environment with a variable number of manufacturing cells. This type of data, which can be prepared off-line, is entered into the static database, whilst the dynamic database holds all of the real-time system and cell specific data. The dynamic database utilises Common Addressed Memory in distributed Processors (CAMP).

The user can perform the building of specific cell control softwares which are later used as the cell supervisors or controllers. The communication module within the SIT, "CellTalk", allows the manufacturing cells to exchange information directly with one another. The communication technique utilised in the Durham FMS is network software and hardware independent. Both the overall activities of the system and the in-cell activities are scheduled and coordinated in real-time by the cooperation of the system scheduler and sequence control within the cell controllers.

The proposed control methodology was implemented on the Durham FMS where the real-time operation of the system events was demonstrated by making the physical elements of a milling cell function as a real system would, whilst at the same time interacting with three other cells which were computer emulations. The system scheduler took charge of supervisory control to synchronise and orchestrate the overall operation of the FMS.

The implemented network for the Durham FMS was built using BBC microcomputers. An area of memory in each BBC was reserved and utilised as network common memory. This network wide distributed memory provided a mechanism for the `CellTalk` software to carry out information exchange among entities in the FMS.

ACKNOWLEDGEMENTS

I should like to express my grateful thanks to my most knowledgeable supervisor, Doctor Clive Preece, for his continuous encouragement, assistance and guidance.

I should also like to mention my gratitude towards the very helpful technical staff of the School of Engineering and Applied Science of Durham University.

STATEMENT OF COPYRIGHT

DECLARATION

No material from this thesis has previously been submitted for a degree at this or any other university.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

AE	Application Entity
ASN.1	Abstract Syntax Notation one
AFMCS	Advanced Factory Management and Control Systems
AI	Artificial Intelligence
AMRF	Automated Manufacturing Research Facility
AGV	Automated Guided Vehicle
AP	Application Process
CAD	Computer Aided Design
CAM	Computer Aided Manufacturing
CAM-I	Computer Aided Manufacturing International
CAMP	Common Address Memory in distributed Processors
CASE	Common Application Service Element
CCSS	Cell Controller Specific Software
CCU	Central Control Unit
CIM	Computer Integrated Manufacture
CNC	Computer Numerical Control
CSMA/CD	Carrier Sense Multiple Access with Collision Detect
CS	Companion Standard
DBMS	Data Base Management System
EFT	Earliest Finishing Time
EMUG	European MAP User Group

EPA Enhanced Performance Architecture

ESPRIT European Strategic Programme for Research and Development in Information Technology

FMS Flexible Manufacturing System

FSK Frequency Shift Keying

FTAM File Transfer, Access and Management

GM General Motors

GT Group Technology

ISO International Standard Organisation

JIT Just-In-Time

LAN Local Area Network

LP Linear Programming

MAP Manufacturing Automation Protocol

MMPM Manufacturing Message Protocol Machine

MMS Manufacturing Massage Service

MPS Master Production Schedule

MRP Material Requirement Planning

NIST National Institute of Standards and Technology

NEMA National Electrical Manufacturers Associations

OR Operational Research

PAC Production Activity Control

PLC Programmable Logic Controller

PPC Production Planning and Control

RF	Radio Frequency
RPC	Remote Procedure Call
SASE	Specific Application Service Element
SD	System Definition
SIT	System Integration Tool
SMM	System Monitoring Module
SQL	Structured Query Language
TDM	Time Division Multiplexing
VIA	Versatile Interface Adaptor
VMD	Virtual Manufacturing Device
WIP	Work-In-Progress

CHAPTER ONE
An Introduction to the Proposed Control Methodology and its Implementation

1.1 Introduction

The last few years have seen dramatic changes in manufacturing. Information technology is no longer limited to the office environment. On the shop floor computers to assist in control, manufacturing and management are emerging. Due to the ever greater number of product types and shorter product lives, new demands are being made on companies in the manufacturing industry. Also, companies are being increasingly forced into rationalising all operational sequences by the demand for a quick and cost efficient adaptation to changing market conditions. Modern computers and production techniques must be utilised to support the development of a product from conception through planning and manufacturing to the successful market sale, in order to achieve the highest degree of flexibility and productivity in a company.

Today this situation calls for manufacturing facilities by means of which small batch sizes can be economically produced with high productivity and quality. The Flexible Manufacturing System (FMS) is the focus of attention for the performance of these tasks. The objective is a computer integrated and flexibly automated production facility for the manufacture of a variable range of products. This requires a combination of hardware, software including an integrated control of production and job scheduling, communications systems, databases and also a dynamic provision and allocation of material, tools, conveyors, clamping fixtures and quality control systems/equipment.

The Flexible Manufacturing System is the current level of applied automation in the field of manufacturing. The application of FMS is time-consuming and expensive, necessitating advanced technical knowledge. A FMS is an automated production system created to achieve various benefits in part (component) manufacture. One such benefit is more beneficial productivity in terms of the unit costs of the components being produced. Another important benefit of FMS is flexibility as regards the component types that can be manufactured. This can be further defined as:

a. long term flexibility - the ability of the system to adapt to totally different component type mixes over the years that the FMS is in use.

b. short term flexibility - the ability of the system to change from hour to hour, or day to day, the component mix being manufactured at any one time.

c. instantaneous flexibility - the ability to reschedule components to alternative machines immediately a machine breaks down.

A FMS can be realised by many configurations of hardware for the machines and controllers and their relevant software. It consists of one or more production cells comprised of groups of automatic work stations (i.e. machining centres) or programmable devices (CNCs) served by a robot or part handling system. These cells are linked by one or more automatic part handling systems all under a production management control computer (overall control system). The overall control system plays a major role in the efficiency and cost effectiveness of a FMS.

Sound strategies that enable the achievement of production requirements, minimisation of inventory levels and effective exploitation of manufacturing resources are required in order to accomplish an overall control system for Flexible Manufacturing Systems. The provision of a vast amount of production data which must be processed by a fast and reliable computer based system is necessary if these strategies are to be generated. Currently, to achieve this in real-time is a challenging subject of research attempted by several organisations [Achatz 1987], [Albus 1981], [Duffie 1986], [Weston 1989], [Alting 1989].

Because reduction in processing times and setup durations have not been translated into a corresponding reduction in inventory levels, little has been gained, since the production of numerical control technology, as regards manufacturing control. The labour content of jobs were apportioned the blame for this until recently. Now however there is a strong feeling that this may no longer be true. Slack production control is at present considered to be the correct recipient of most of the responsibility for the problem. To overcome this therefore, there is an urgent need for a control methodology.

There are two major obstacles to a successful FMS installation both of which affect the overall control system in a FMS. The first obstacle is the communication problem. In a FMS, every programmable device (machine) must be intelligent enough to communicate with others in the system, since most manufacturing requires multiple machines. Therefore, communication is very important to provide the coordination needed to match the sequence of activities within a FMS. Vendors tend to make programmable devices with their own proprietary protocols, complicating communication within the network. To solve this problem, protocol translating software is often developed to run in the vicinity of the devices.

Secondly, there is no unified FMS strategy which is applicable across the board covering all types of manufacturing. Since each manufacturing operation uses different programmable devices and different communication protocols, it is very difficult to make a unified FMS approach work. To overcome these problems several bodies, both in Europe and USA, have put much effort into the proposal of an architecture for shop floor control and management.

1.2 Control Architecture in Automated Manufacturing

There are two approaches which have been proposed to control shop floor activities: hierarchical and heterarchical.

The most famous hierarchical models are proposed by the Automated Manufacturing Research Facility (AMRF) of the National Bureau of Standards (NBS) in USA [Albus 1981], [Brown 1988] and the Advanced Factory Management and Control System (AFMCS) of the Computer Aided Manufacturing International (CAM-I) [Bunce 1988]. Both models are very similar and divide the manufacturing system into several levels according to their functions and information flow. The levels are introduced to reduce complexity and limit responsibility. The control modules for these systems are organised into a hierarchy, i.e. a tree structure. Each module has only one supervisor, but may have several subordinates. Control modules in the middle of the hierarchy are both supervisors and subordinates. The upper level requirement is decomposed at each control level before a command is issued to the next lower level. The planning horizon becomes smaller at the lower levels of control. There are two major principles which are implicit in these hierarchical structures.

a. decisions are made at the lowest possible level;

b. control resides at the next highest level.

The disadvantages of a hierarchal control system include; no peer-to-peer communication is allowed; the system heavily relies upon the production schedule prepared by the higher levels, a crash at some levels will bring the entire system below to an abrupt halt.

An alternative to hierarchical is a heterarchical control architecture [Duffie 1986]. In such a control structure there is no supervisor and no direct control. All entities are equal and take part in a negotiation process to plan, schedule and manufacture products [Duffie 1987]. Control decisions are reached through mutual agreement and information is exchanged freely among the participant entities. There is no need for a scheduler and production of parts takes place by the cooperation of intelligent entities. The disadvantage of such a system is that free data exchange is imperative at all times and can lead to very slow decision making. Furthermore, a misread communication between entities may cause a system deadlock.

NBS and AFMCS are not the only organizations which have been addressing the problem of shop floor control and management; there are other development efforts given by and including the ESPRIT project 477, ESPRIT project 418 and University College Galway. There are also other projects [Graefe 1989], [O'Grady 1989] but only a few are selected for a brief description.

The ESPRIT project 477 was carried out under the "Control Systems for Integrated Manufacturing: The CAM solution" title [Esprit 1989]. The objective of this project was to design, develop and test the software modules required for the production activity control of small batch manufacturing. The aim was to close the loop between production planning and execution, reducing human intervention and reaction time as much as possible and relying on automatic shop floor data collection.

The ESPRIT project 418 was executed under the title of "Open CAM System Allowing Modular Integration into Factory Management of a Workshop Structure in Functional Cells with Various Levels of Automation". The objective of this project was to develop a CAM system with an open architecture which could integrate and monitor on-line batch production planning and control activities, such as shop floor control, handling and quality control, for the manufacture of mechanical products.

A Research team at the University College Galway in the Republic of Ireland has been active in areas such as job shop control decision making and scheduling in a batch or job shop production environment. The objective is to resolve the problem of real-time planning and control on the shop floor.

However, none of the above research works combines the hierarchical control structure with the heterarchical control system. Since it is believed [Nahavandi 1990] that a control system could take the best points of the these two dissimilar control structures, a hybrid control architecture was proposed for the Durham FMS. This control structure combined hierarchical and heterarchical control methodologies and was implemented on an experimental FMS rig. The control system for the FMS was divided logically into three levels of hierarchy (master, cell and equipment) with the Master at the top level. The system and jobs definition took place at the master level and when an order was put into the Master to process a job, commands were sent to the lower levels by the Master to process the job. Entities at the cell level decomposed the commands before making decisions on how to manufacture the parts in their cells. Thus exhibiting the hierarchical behaviour of the model.

The heterarchical behaviour of the model was shown by the ability of entities to exchange information with their peer (other cell controllers) and other entities (i.e. Master). Cell controllers only needed to obtain an order from the Master on what job to be processed and in what quantity and from then onwards they would exchange information with one another and from time to time get a system overview from the Master to make decisions locally until the job was successfully processed. Therefore the decision making was distributed among the entities but at the same time since the Master had an overview of the system as a whole, entities could refer to the Master to gain a better view of what was going on without too much exchange of information amongst themselves.

To apply the proposed hybrid control model to the experimental FMS rig a System Integration Tool (SIT) was designed and developed to exhibit the flexibility and vendor independency of techniques proposed and developed.

1.3 The System Integration Tool

The idea of implementing a hybrid control system on the Durham FMS has been put into practice with the System Integration Tool (SIT). This collection of designed and developed software modules is comprised of four main building blocks, Fig. (1-1). The manufacturing database module has two parts, one holding system static and the other system dynamic data. The Static database holds all of the information regarding the system, resources and jobs. It allows the user to define its own manufacturing environment and also define jobs. The dynamic database is responsible for the holding of information on job processing, during a period of part production. For example, information such as the status of devices in a cell is held here. The dynamic database utilise a Common Address Memory in distributed Processors (CAMP). As this allows rapid access to data by various entities throughout the FMS.

The second element of SIT is the communication module. It allows all of the entities to communicate with each other via a Local Area Network (LAN) in the

FMS. The LAN in the Durham FMS is Econet which is similar to an Ethernet system. The developed network communication protocol, "CellTalk", takes advantage of the direct memory access of the different entities' CAMPs. Cell controllers being able to communicate directly with one another, will demonstrate the heterarchical nature of the overall control system.

The third building block of the SIT is the cell control builder module. Since the signal required to control a manufacturing cell is highly application and device specific, the control command sent to the device controllers in each cell has to be translated and formatted in a suitable form for that entity. Hence, a software was developed which allowed the user to insert the device specific code for each control command. For example, a cell controller, to start the operation of a programmable device (milling, turning, washing, etc.), each device controller may require a different control signal for the same function. Therefore, generic routines were defined for each control function (start, stop, reset, etc.) where the specific control signal was included. In this way, one Cell Controller Specific Software (CCSS) was built for each cell controller. Figure (1-2) illustrates the tasks which a CCSS has to perform during and at the end of a part production period.

Finally, the fourth and last building block of SIT is the Scheduling module. During a period of part production, activities within the FMS must be scheduled. This is done by two collaborating entities one at the master and the other at cell levels. The entity in the master level utilises the scheduling algorithm to synchronize the Intra-cell activities while the scheduling and sequencing of in-cell operations is carried out by the different cell controllers. This method was adapted to introduce some level of autonomy to the cell controllers while they, at the same time, exchange information with the entity at the master level for an overall view of the operation, and exchange data with other cells for specific data required to perform certain tasks. This is achieved by the overall scheduler, dispatching parts for production to the different cells in a user defined sequence. Once a cell is in possession of a part, it exchanges information with other entities to obtain the required data before processing the part.

For example, parts leaving a cell may be placed on a pallet. The receiving cell will then need the location and orientation of each part being placed onto the pallet by the previous cell robot. Cells can utilise the `CellTalk` to obtain this information from one another. Hence leading to the term "heterarchical" control.

For the user to apply SIT to the FMS correctly, various system parameters have to be defined at the master level through the execution of the MASTER program which is a menu driven module. Figure (1-3) illustrates some of the options available in the MASTER program. The user must first define the manufacturing environment before starting to build the CCSS for different cells. Next, jobs can be defined together with the sequence in which the parts are allowed to visit the cells. Jobs may then be executed for production and while parts are being manufactured in different cells, a system and job status report can be obtained. These reports are collected by the system and archived for historical analysis. All of the system activities are also recorded in real-time and can be viewed graphically for off-line system analysing. A further description of each of these options is given in the subsequent chapters.

Applying the System Integrating Tool (SIT) to the FMS was carried out to achieve a preset goal for this research.

1.4 Research Objectives

Although in recent years several industrial packages have become available for the control and monitoring of manufacturing processes [PC Hardware and Software Guide 1990], many small companies cannot justify the implementation of such systems to their plants. Furthermore, once such packages have been implemented, they tend to make the company vendor-dependent and often do not allow for easy integration of modules from different vendors.

The objective of this research was to overcome such problems whilst at the same time offering a low-cost solution. Figure (1-4) outlines the objectives of this research which are divided into five categories, described in turn. A control system was required to allow for an effective overall control of a manufacturing environment. A hybrid control architecture was designed to cater for this need. The Durham FMS was to be used as a test-bed to demonstrate, so that the practical aspects of the developed software were not overlooked. The software was to be designed in such a way as to facilitate for easy system redefinition and expansion.

Another objective was to develop a system which allows equipment from a wide variety of vendors to be able to integrate into one system. Also, since the cost of a baseband network is lower than that of a broadband, implementation on a baseband system was set as an objective. Finally, in order to make the controllers of various devices in the FMS communicate with each other irrespective of their make and model, the design and development of a network independent communication technique was set as an objective.

Work carried out during this research targeted achieving these objectives and this thesis attempts to forward a comprehensive report on the proposed ideas and designed and developed modules. The structure of the thesis can be considered as a logical division of the work into several modules which are the representative of certain activities in the SIT software.

1.5 Thesis Structure

The thesis is comprised of seven chapters. Chapter one gives an introduction on the subject of the research, current methods in use and the proposal of what is thought to be a better solution to the problems involved in manufacturing control. Chapters two through five describe different modules of the System Integration Tool (SIT) which is utilised as a means of applying the proposed control methodology to the experimental FMS.

Different types of control systems together with their advantage and disadvantages are described in chapter two. The physical setup of the Durham FMS is examined to highlight the need for a vendor-independent control strategy. Next a hybrid control architecture comprising hierarchical and heterarchical structure is proposed. Finally, the importance of the control of programmable devices without human intervention (remotely controlled) in a FMS is shown. A programmable device (CNC miller) of the FMS was chosen as an example and a method of how such a relatively non-flexible CNC machine could be adapted for a FMS environment (to operate remotely under the control of supervisory computer) is demonstrated.

Chapter three looks at how information should be organised in a highly automated environment such as a FMS and what advantages could be obtained by this. A manufacturing database is designed and developed, comprising of two parts, static and dynamic databases. Each of these are examined and the type of system information which is contained in them is also given. Next, a simple drawing package developed for the design of simple shapes is described as this shows how the information relating to a part design can be presented and for the design information to be understandable to a programmable device the information is post processed (for the CNC miller). It is also mentioned that this software is developed as an example for representing the design and associated table code for CNC machines and therefore is application specific. However, the system has the potential to receive design information from other sources.

A Production Planning and Control (PPC) system is described in chapter four. Planning and control of a FMS is highlighted and a mathematical model which computes the FMS cell capacity check is described. Next, the scheduling of shop floor activities is explained. Following this the real-time scheduler which coordinates the Durham FMS inter-cell activities is described. The intra-cell activities are coordinated by the cell controllers, therefore the functions of cell controllers together with their operation sequencing are described here.

In chapter five the communication problem in manufacturing is described. The idea of an open system and its advantages is explained which is then followed by a description of Local Area Networks (LAN)s. Next, the General Motor's solution to the problem of shop floor communication is explained. Following this is the LAN utilised in the Durham FMS and its advantages and disadvantages over other types of networks. A communication technique and the service primitives developed for the Durham FMS is then described. The network independency of such a communication technique is also highlighted.

The collection of system information and monitoring is described in chapter six. Different ways of information representation are discussed. A description of the on-line report generation of the system scheduler is given. This is followed by the on-line recording of system events and the graphical representation

(System-wide scheduling chart) of the system scheduler's activities. Next, all of the obtainable information from the chart is presented. It is then described how the up-load file is prepared and generated in the system.

Finally, the discussion and conclusion chapter states what objectives have been achieved. The advantages and disadvantages of the implemented ideas are discussed and also suggestions for further work are given.

There are five appendices which provide further information on the designed modules. For example, Appendix D describes the detailed programming of the Cell Controller Specific Software (CCSS).

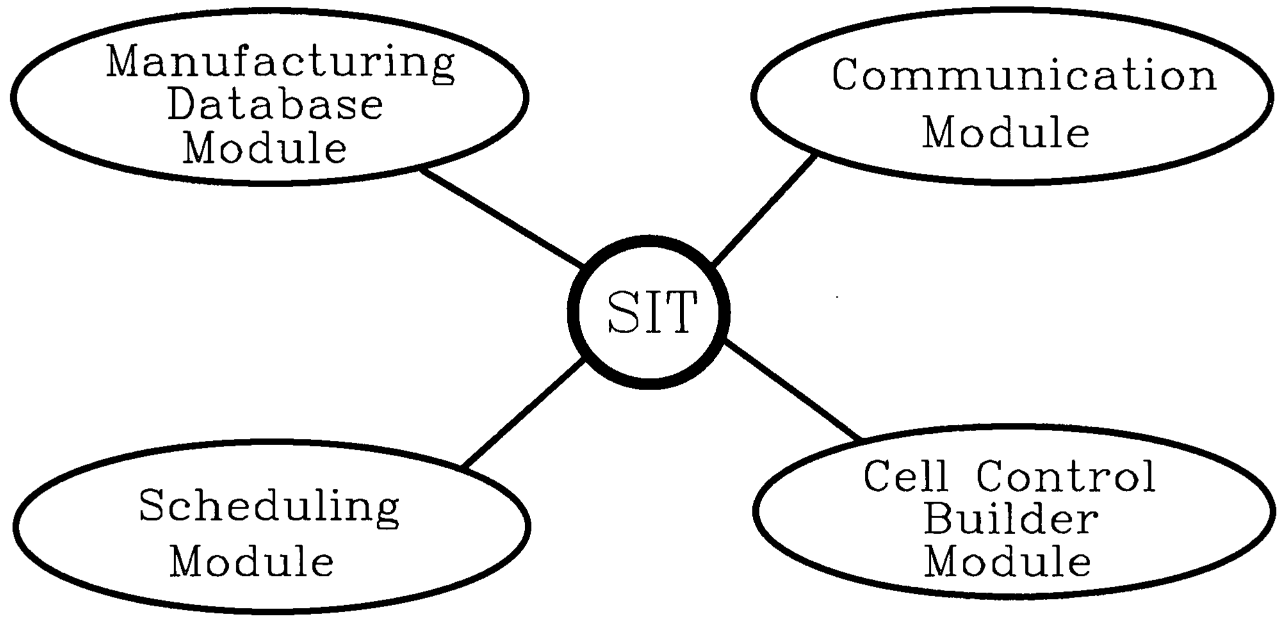


Figure (1–1) The building blocks of the System Integration Tool (SIT).

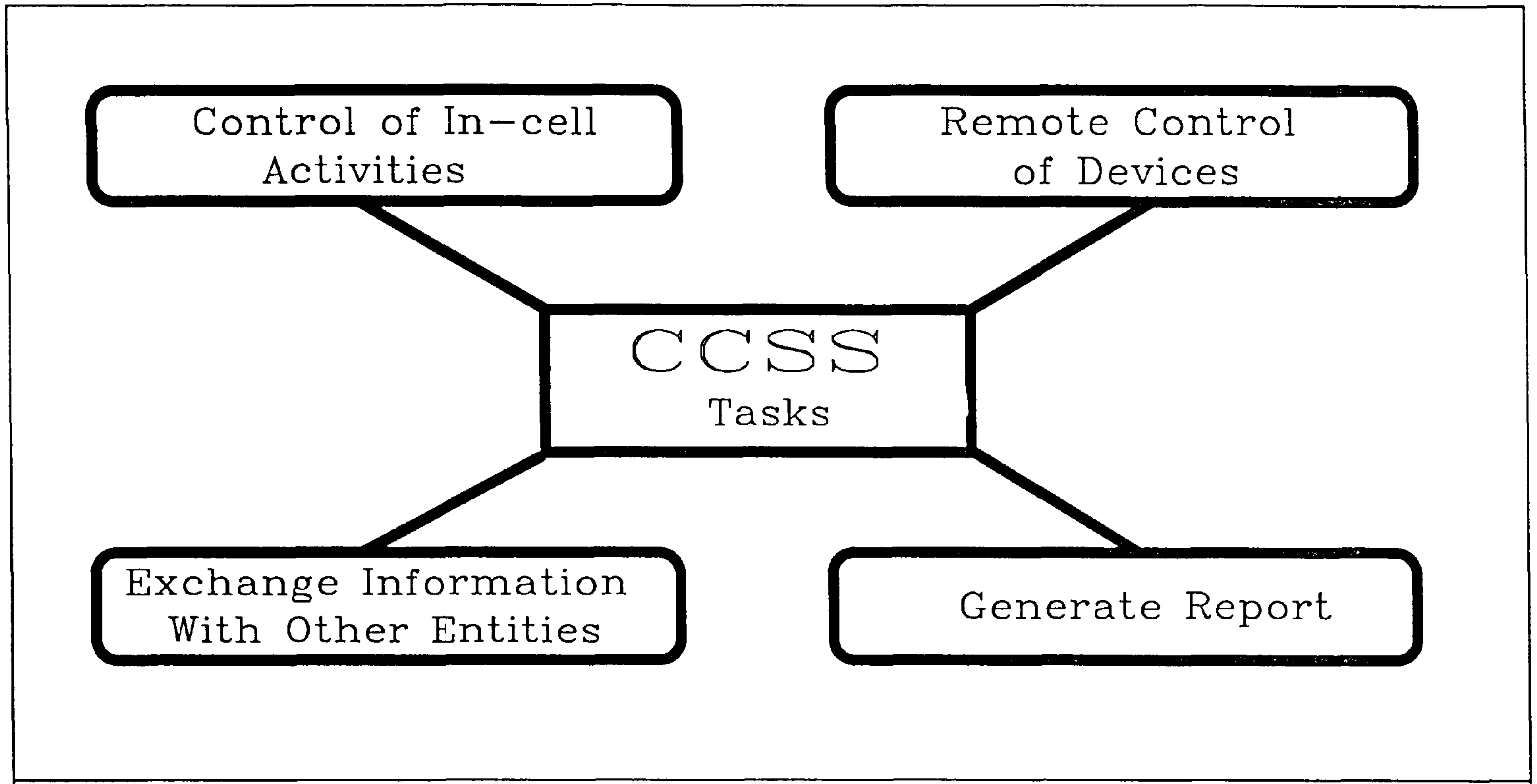

Control of In-cell Activities
Remote Control of Devices
CCSS Tasks
Exchange Information With Other Entities
Generate Report

Figure (1-2) Tasks performed by a Cell Controller Specific Software (CCSS).

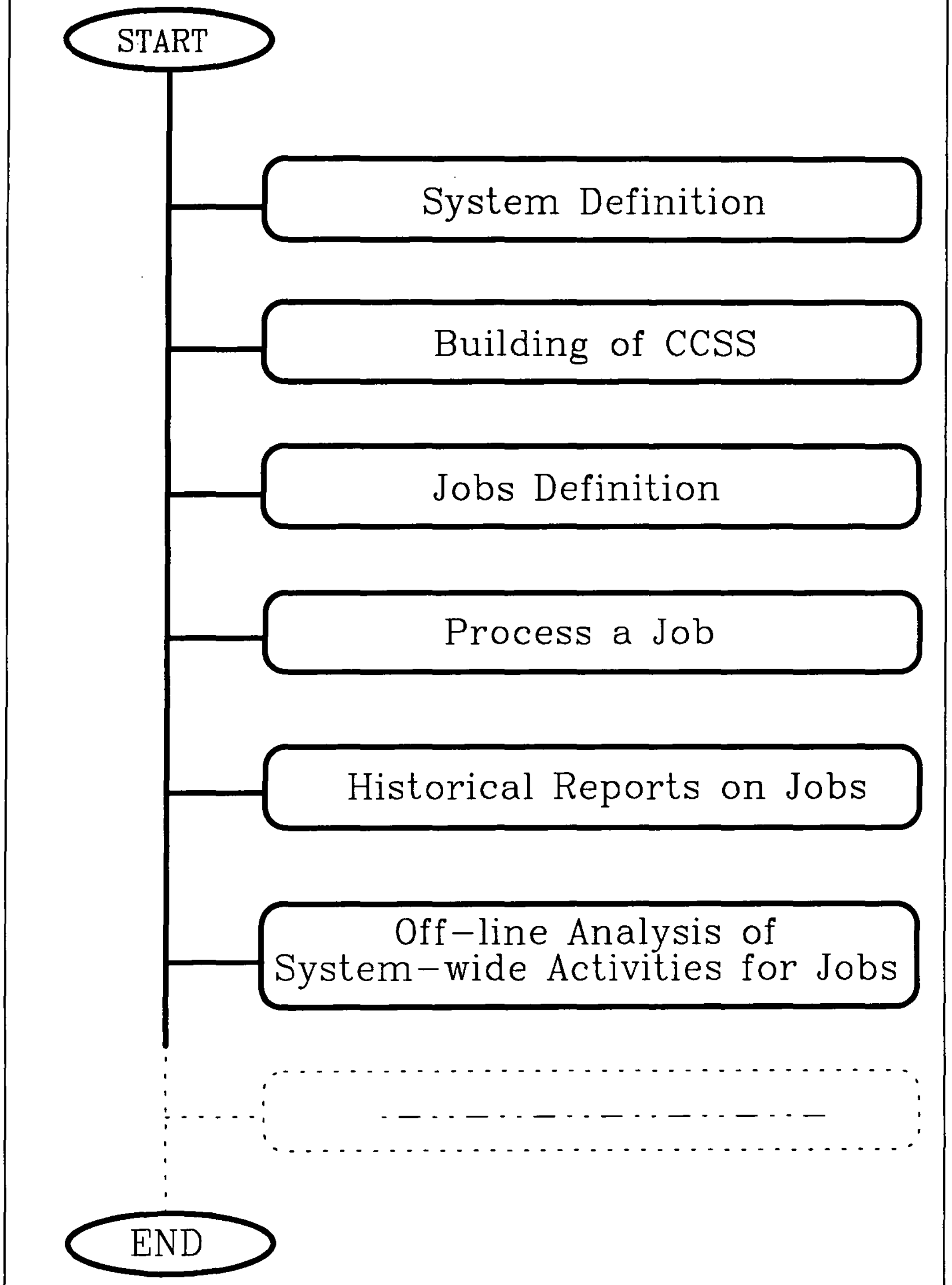

START
System Definition
Building of CCSS
Jobs Definition
Process a Job
Historical Reports on Jobs
Off-line Analysis of System-wide Activities for Jobs
END

Figure (1-3) Options available within the MASTER program.

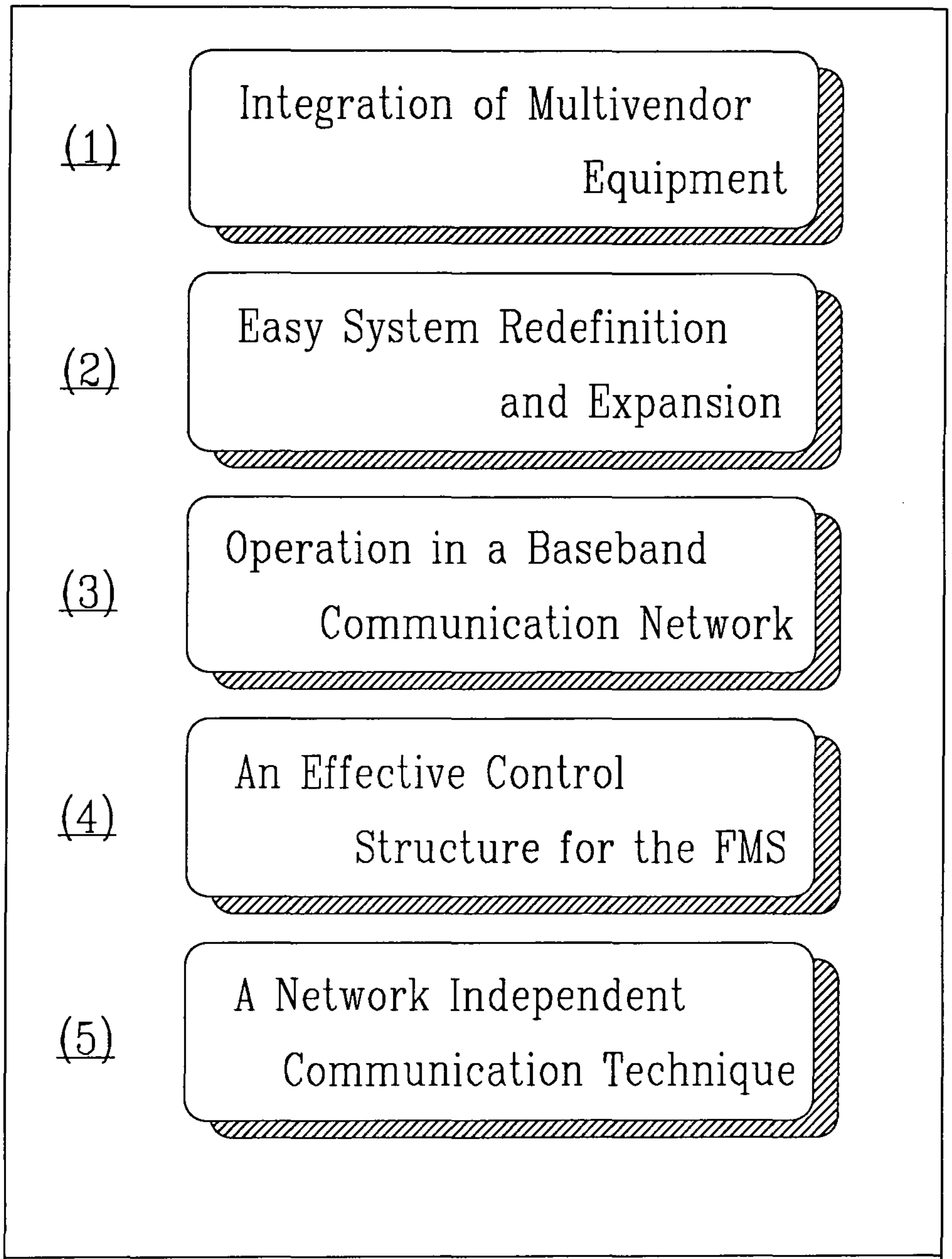

(1)
Integration of Multivendor Equipment
(2)
Easy System Redefinition and Expansion
(3)
Operation in a Baseband Communication Network
(4)
An Effective Control Structure for the FMS
(5)
A Network Independent Communication Technique

Figure (1-4) The research objectives.

CHAPTER TWO

Control Organisation in Flexible Manufacturing Systems

2.1 Introduction

Flexible Manufacturing Systems (FMSs) consist of a variety of automated manufacturing equipment linked together with a communication network and a part transfer system. Production planning, Computer Aided Design (CAD) and other management and engineering support systems can be integrated into the FMS. Flexible Manufacturing Systems typically are designed to manufacture a variety of part types in jobs with batch sizes of down to one. As a result FMS may be able to produce complex parts with higher quality and short throughput time at a reasonable cost.

Short throughput times with frequently changing batch sizes require machines and systems which can quickly be reset and reprogrammed. This makes the control of a FMS very complex, with a large number of interrelationships between machines, tools, parts and materials. The problem is compounded by the requirement for short manufacturing lead times and the need to provide detailed control of all manufacturing operations.

This chapter addresses the problem of designing control architecture for Flexible Manufacturing Systems. Section 2.2 describes the current manufacturing trend and cellular manufacturing. Next, the physical elements of the experimental FMS is described. This information is given in section 2.3 which is then followed by section 2.4 centring the discussion on a different control structure for the overall control of a FMS. In the next two sections 2.5 and 2.6, a closer look is given to two control architectures, hierarchical and heterarchical, which have been the subject of some research, to overcome the problem of decision making and control in automated manufacturing systems. Section 2.7 highlights the advantages and disadvantages of the two control structures and their comparison. This is then followed by section 2.8 proposing a hybrid structure for the control of the Durham FMS. Finally, section 2.9 describes the need for an interface module to enable the implemented control architecture to function fully and in real time.

2.2 Current Manufacturing Trends

As automation brings higher productivity to the manufacturing industries, attention must be focused on the selection of the automated system. Such systems must be flexible enough to accommodate short and mixed production runs, combined with a requirement for very short product-development and manufacturing lead times. Prototypes are needed not in weeks but within hours, and modification to the prototypes must also be available within hours.

A Flexible Manufacturing System (FMS) can provide the basis for a versatile production and assembly operation whose control is provided by an integrated computer system [Nagarkar 1988], [Tchijov 1989], [Adams 1988], [Nordsten 1989]. One of the currently accepted FMS structures is to divide the manufacturing and assembly process into cells [Wemmerlov 1989], [Martin 1989], [Al-qattan 1990]. This method of cellular manufacturing is sometimes referred to as one of the best strategies in use to implement the Just-In-Time (JIT) production [Helmut 1988].

A typical cell may be comprised of two or more programmable devices served by a robot. Each manufacturing cell can provide a range of potential operations which can be selected by the controlling computer. Note that at present some production cells may require the intervention of a human operator as opposed to being fully automated cells. These may fall into the category of manned cellular manufacturing, where the objectives include reduction in material handling, elimination of Work-In-Progress (WIP), improvement of scheduling and reduction of overall throughput time and, most importantly, full utilisation of the human resources [Black 1988]. Here the human operator serves a multifunctional purpose and is decoupled from the machines, so that the utility of the human operator is no longer tied to the utility of the machine. As a result there may be fewer human operators in each cell than machines.

Alternatively, in unmanned cellular manufacturing, utilisation of the equipment becomes more important since the flexibility offered by the human operator is eliminated by robots and other programmable devices. Although Flexible Manufacturing Systems will incorporate the idea of cellular manufacturing, it is expected that the cells within a FMS be unmanned (i.e. fully automated).

Many manufacturing companies build their FMS from cells which are capable of operating stand alone. A FMS may be comprised of one or two cells at the beginning, but there must exist the potential for capability of integration of further cells [Kovacs 1988]. This brings the problem of compatibility between products from different vendors at cell or at system level, requiring modification every time new equipment from a different vendor is added. As a result, most of the existing installed FMS plants comprise a series of integrated units from one supplier.

The Durham University FMS is a good example of such a system where multi vendor products are utilised to make the operating elements of different cells. Before discussing the different control strategies which could be employed in the control of such a system, the hardware elements of this FMS are explained.

2.3 The Durham University FMS Rig

An experimental Flexible Manufacturing System has been set up at Durham University. The available resources include; a CNC milling machine, a CNC lathe, five robots, a sawing machine together with the associated part feeder mechanism, a part transfer system, and a vision system, Fig. (2-1). The raw materials are stocked in the warehouse and the finished parts are transferred to the finished goods store. Also, part of this store is dedicated to the reject parts.

In order to reduce a machine's idle time, and to reduce the chances of a machine waiting for a part rather than a part waiting for the machine, each cell is equipped with two part buffers. Parts, on arrival at a cell, are deposited on the cell input part buffer, and, after processing, the parts are transferred from the machine's table to the cell output part buffer, to await transfer from the cell. To keep the inventory low, the size of these buffers is kept small.

The part transfer between the cells is performed by a device (pneumatic trolley) which provides part transfer from a cell to any other cell selected by the control functions. In this way, the order in which the cells have to perform operations on a job is not confined to a particular fixed sequence for all the jobs, hence increasing the overall system flexibility. The SIT system software was tested on the experimental plant in order to ensure that practical aspects of its operation were not overlooked. In defining the operations of the experimental FMS the following assumptions were made.

The available resources were to be utilised in such a way that the manufacturing tasks performed by each cell were defined by their function. The FMS is divided into several cells, each defined as executing a certain type of task, for example, an assembly cell will be unable to perform a milling operation. Each cell is further defined in more general terms as having one or more programmable devices served by a robot. Hence, the turning cell comprises a CNC lathe together with a robot. With this definition of a cell the FMS can be expanded more easily since each cell performs its tasks autonomously, although its operation may rely on information exchange between itself and other cells.

Having established the autonomal behaviour of the cells, each cell can then be further partitioned by functions into stations, since the functionality of cells is independent of each other. Work stations could then receive orders from their cell controllers to process a part but how to go about it would be the responsibility of the station controllers. Therefore a FMS can be set up as simple and small with the potential for easy future expansion.

In the experimental FMS, each cell possesses a microcomputer which acts as a cell controller to which various programmable devices are linked by means of point-to-point connection. Each cell controller is linked to a backbone local area network using Ethernet-like (Econet) protocols. The networks provide the means for all the cell controllers to exchange information with the system scheduler and each other.

The effective operation of a FMS requires an efficient control structure to operate the system in the desired manner. The design of the control structure is therefore critical to the overall performance of a FMS.

2.4 Control Methodology for a FMS

The functionality of a FMS is often reflected through the way in which it is controlled [Greenwood 1988], [Maimon 1987]. Installing computer controlled and programmable devices in a manufacturing environment does not necessarily bring the ultimate system flexibility anticipated by the system managers and system implementors, unless the correct strategy towards the control of the system as a whole is taken [Anstiss 1988].

The overall control strategy [Maley 1988] plays a major role in utilisation of the system to its best, and it should allow for easy expansion of the system entities. Entities here are defined as the system hardware, software or a combination of both modules. The problem involved with the overall control of entities in a FMS is far greater than the problems involved with the control of individual entities. This is because the complexity of system control grows rapidly with the number of individual entities.

As demand increases for high reliability and performance of FMS control, the need for a system with efficient data processing and real-time error recovery becomes unavoidable. Centralised and distributed control are two fundamentally different methods of controlling the FMS.

Flexible Manufacturing Systems with centralised control architecture are characterised by the existence of a Central Control Unit (CCU) which may be a host computer [Achatz 1987] together with a number of programmable devices built around it, Fig. (2-2a). The arrangement of the FMS could be in the form of a star and the central control unit estabilishes a strong relation of master-slave to the devices attached to it. Decisions are centralised at CCU and also made by it. With this type of control structure it is guaranteed that the central control unit can have access to all the programmable devices in the system, thereby enabling it to execute a production schedule. A system with hierarchical control structure (discussed in the next section) could also exercise this type of centralised decision making. Decisions are then made at higher levels before being passed down to lower levels to be carried out. The advantages of systems with a centralised control structure are:

a. broader view of the system. It is possible for better decisions to be made by the CCU because of its broad view of the system. This broad view is due to the fact that it has the status feedback from all entities (programmable devices).

b. the problem of complex inter activity between decision makers is eliminated because there is a central decision maker (i.e. CCU).

There are a number of disadvantages with centralised control systems which overshadow the advantages. These being;

a. the CCU may have to handle large amounts of data on each status change in the FMS and also the volume of data becomes enormous with larger FMS's.

b. the CCU has to make decisions on every activity within the FMS. The level of detail required can be overwhelming for any medium or large scale FMS and some congestion can occur at the CCU leading to slow response times.

c. the entire FMS will come to standstill in the case of a central control unit failure.

d. a decrease in reliability results as the size of an installation increases and the complexity and software overheads increase.

e. a backup computer is often used to eliminate the possibility of total system crash. This will introduce extra complexity and cost as further monitoring devices must be incorporated to detect a system failure.

These disadvantages have prompted attention to be focused on more decentralised decision making. This alternative to the above control structure is a FMS with distributed control architecture [Wang 1989], [Villa 1988]. Geographically distributed intelligent modules (processors) are connected to a LAN to share the processing load of the control system. There are no master-slave relationships or central supervisory. Supervisor activities are performed by the distributed processors as a group in an orderly and cooperative manner.

Work in the Delft University of Technology [Bakker 1988] demonstrates the operation of a FMS with a distributed control architecture, Fig. (2-2b). Each programmable device is interfaced to an intelligent module, which controls the sequence of its operations. The intelligent modules negotiate with each other over the LAN to arrange which operations are to be performed by each programmable device. There exist no production schedules and the operations are allocated to the devices during the actual period of part production, not before. The intelligent modules in effect act as an agent on behalf of the programmable devices and are responsible for the availability of the materials, required tools and programs. Since the intelligent modules do not attempt to optimise the devices' productivity by changing the sequence of operations, this may count as a disadvantage for such control systems. The operation and control structure of a distributed FMS resembles, in many ways, the heterarchical control system, which is discussed in section 2.5.

One alternative way of considering the overall control problems for a FMS and to provide solutions is to either introduce a hierarchical [Albus 1981] or heterarchical control structure. Introducing a hierarchical control structure would allow control problems to be partitioned into manageable modules regardless of the complexity of the complete structure [Brown 1988], [Golenko-Ginzburg 1988]. On the other hand, a heterarchical control structure gives equal rights to all of the entities within the FMS and decisions are reached through mutual agreement between the entities. The following describes each control structure in turn.

2.5 Hierarchical Control Structure

Since the control of a FMS is a very complex problem with a huge number of interrelationship parameters between resources, jobs and machines, one approach which may accommodate such complexity is that of breaking the control into a hierarchy [Albus 1981], where each level of hierarchy has narrower responsibility. As the hierarchy is descended, the time period considered shortens whilst the level of detail considered increases [Brown 1988], [Golenko-Ginzburg 1988]. The common names for these levels include factory, shop, cell, area, and equipment. Hierarchical control for manufacturing industries is often found under a wider area of Computer Integrated Manufacture (CIM) [Kusiak 1988].

The design of a hierarchical control structure is based on three guidelines:

a. levels are introduced to reduce complexity and limit responsibility;

b. each level has a distinct planning horizon which decreases as the hierarchy descends;

c. control resides at the lowest possible level.

The application of these guidelines has led to the design of a variety of different architectures. The differences include the number of levels and functions assigned to each level. There are two separate bodies which have made a major contribution towards the proposals for hierarchical control of automated manufacturing systems. These are the Computer Aided Manufacturing International (CAM-I) with its Advanced Factory Management and Control System (AFMCS) model and the Automated Manufacturing Research Facility (AMRF) of the National Bureau of standards (NBS) in the USA [Simpson 1982] with its own architecture. The work by these organisations has resulted in the development of two similar standards. Both standards divide the complex planning and control functions of a manufacturing system into four levels, Fig. (2-3). In the AMRF model the levels from four to one are: Facility, Shop/Cell, Workstations and Equipment respectively, and in the CAM-I model these levels are: Factory, Job shop, Work centre, Manufacturing unit. In both models, level 4 is the highest level of control where the system-wide control is applied by the management executives. The only difference between these two models is that in the Shop/Cell level (which is sometimes divided into two separate levels) of the AMRF model a concept of "Virtual Cell" is introduced. This is an abstract cell for planning and control which allows the actual size of real cells to be varied.

In the CAM-I model the factory level is concerned with the determining of end product requirements, product structure definitions and individual shop capacities and capabilities and is the top level of control. The level below this is the job shop level which takes end product production rates and explodes them into processing operations. Shop order events can then be scheduled and commands passed to the work center level. This generates and schedules detailed task requirements. These requirements are passed to the lowest level of the CAM-I hierarchy, which is the unit level. This breaks the tasks into subtasks which are then implemented.

In the AMRF model there exists the facility level at the top which includes process planning, global management and information management. Below this is the shop level which manages the coordination of resources and jobs and the operation of individual manufacturing cells on the shop floor. The grouping of parts belonging to certain jobs using the Group Technology (GT), [Hyer 1989], is one of the functions conveyed at this level. The concept of a virtual manufacturing cell is introduced at this stage. These virtual manufacturing cells contain machines which are grouped together in a dynamic fashion. In addition tasks such as allocating tooling, jigs, fixtures and materials to specific workstations are the responsibility of this level. Below the shop level is the cell level, where the cell control system schedules and controls jobs, material handling systems and tools through the cell. Jobs at this point have already been divided into groups and, depending on their similarities, are allocated to each cell.

The next lowest level is the workstation level which is responsible for the coordination of activities within a workstation. This includes the arrangement of sequence of operations for successful completion of jobs allocated to a particular cell controller system. A workstation may consist of programmable devices such as an NC machine tool, a robot, a control computer and material buffer storage. There is equipment at the lowest level of the control hierarchy which consists of the controller for individual resources and programmable devices.

For a Flexible Manufacturing System utilising a hierarchical control structure, the control and planning functions may be divided into several levels. Communication will play a major role amongst the various levels since information gathered at one level is used in another. The volume of data, and the amount of data processing will vary in each layer of hierarchy, and it is good practise to design the control system in such a way that only the most relevant data is passed on to the next level up in the hierarchy. Entities at lower levels require a shorter response time, as opposed to the ones at higher levels which are controlled by the management executives.

Each level will receive upper level requirements, and the real-time feedback information from the level below, before making decisions and issuing the control commands to the next level below [Mclean 1988]. The control commands are then decomposed to the appropriate tasks, before being allocated to the various entities in that level. Each level is able to operate autonomously to achieve its own objectives, executing the orders from the upper levels.

In a strict hierarchical control structure, entities at each level can be interconnected to the immediate level above by means of point-to-point connection, without requiring a factory LAN, Fig. (2-4). However, this method of interconnection will restrict the system expansion and prove an inefficient utilisation of useful common resources within the manufacturing environment. Therefore, installation of a system-wide LAN would be advantageous.

An alternative to hierarchical control is a non-hierarchical control structure (heterarchical), [Duffie 1986] which takes a different approach to the overall control problem.

2.6 Heterarchical Control structure

The implementation of a hierarchical control system means that a crash at some level can cause the entire system below to come to an abrupt halt. As a result researchers have looked for an alternative control system to eliminate this deficiency [Duffie 1986]. In this approach there is no supervisor, hence no direct control. The autonomous intelligent entities interact to satisfy their needs. All of the entities are equal in the negotiation process to obtain services, and cooperate with each other to obtain mutual satisfaction. If cooperation is used as a control strategy then devices must meet the following requirements; firstly, they should be able to operate independently of the rest of the system, secondly, while being an effective integral part of the system, at the same time should be able to refuse requests for services from other devices.

In the heterarchical system the modularity of the system control is improved and global information is reduced by locating decision-making where the data originates, consequently, the source code for the control program in such systems are smaller and simpler [Duffie 1987]. Another great advantage offered by the heterarchical system is in its fault tolerant approach towards faulty machines, since faulty machines simply will take no part in the negotiation process with the rest of the system. However, the presence of a LAN in a system is absolutely vital since the intelligent entities use the LAN to arrange transactions with other entities, Fig. (2-5).

Heterarchically controlled manufacturing systems are constructed in such a way that each part and workstation is programmed as an intelligent entity that uses the communication network to arrange transactions with other entities. Intelligent workstation refers to that which is able to execute machine coordination, planning and monitoring. Furthermore, the ideal system of the future will be the one which carries out the aforesaid tasks at the machine itself from a product description.

Workstations may hold part programs for NC machine tools and information on tool management, production capacity, processing capability and maintenance records. Parts and workstations are then able to exchange information relating to the processing capability, historical quality characteristics, current load and estimated completion time for the required operation, type of material and type of operation to be performed.

Typical operation of a heterarchical control structure is as follows; Activities commence when a part broadcasts a request repeatedly on the network for a particular type of machine until a response is received from a free machine. The part will then reserve that machine before making arrangements with a robot for its transportation and transmission of part data to the reserved machine.

Since both hierarchical and heterarchical control systems have advantages and disadvantages, a comparison may be necessary.

2.7 Comparison of Hierarchical and Heterarchical Control Structures

Since the overall control system for many FMS is only two or three levels deep, the comparison here will be based on systems with the most popular levels i.e. the cell, workstation and equipment levels. Figures (2-6a) and (2-6b) illustrate a manufacturing cell organised by hierarchical and heterarchical control structures.

In the hierarchical system there is a strong master-slave relationship between the levels. High level goals are successively decomposed by lower level control modules until a sequence of coordinated primitive actions is generated. These primitive actions will be simple machine level commands which can be executed by the programmable devices. In such systems sophisticated planning and scheduling tend to be done only at the higher levels. Lower levels typically are left to execute one instruction at a time. This means that there are, in fact, only two distinguishable levels. One to do the decision making and one to do the controlling. The others merely reduce the information handling and coordination performed by a single computer. A typical example of such a hierarchical control system is the AMRF.

In the hierarchical system, each controller on the same level is connected to the controller in the level above. For a controller to find out the status of the others on the same level, it has to go through its superior level. As a result, no entity can communicate with its peer entity without going through their superior level first. With the heterarchical system this problem is overcome, since all the entities are intelligent and use the network communication as a channel through which to talk directly.

The hierarchical control system heavily relies upon the production schedule, which is prepared earlier by the higher levels in the control hierarchy, whereas in the heterarchical system there is no need for a scheduler, as the production process takes place by the cooperation of intelligent entities (parts and machines).

The expansion of the system and its resources in heterarchical systems is simple. It only requires the addition of extra entity programs to the network as opposed to the hierarchical system where the overall control program has to be edited. Also in a hierarchically controlled system since no negotiation process takes place among the entities, they (entities) do not require a high level of intelligence, as opposed to those of the heterarchical system.

In the hierarchical system, if an entity fails to function correctly, it will disable the functionality of all the entities below it, unlike the heterarchical system where the failed entity will simply dissociate itself from the rest of the system allowing, wherever possible, the rest of the entities to achieve their goals without it. However, in heterarchically controlled systems there exists a fear of system deadlock, to a large extent from a message ambiguity. In addition free data exchange and negotiation is imperative.

To determine a performance comparison of manufacturing cells with the hierarchical and heterarchical control structure, an experiment at the University of Wisconsin-Madison [Duffie 1986] has resulted in the compilation of the following information.

The lines of source code and the cost towards its development for the hierarchical cell control was nine times greater than that of the heterarchical. The machine utilisation in the heterarchical system was lower than hierarchical, since scheduling is performed using a cooperative communication protocol which consequently slowed the algorithm. The memory requirements in the heterarchical system was seven times less than that of the hierarchical. This was due to the fact that the scheduling algorithm is more memory efficient and functions as the cooperation of distributed entities. The average CPU utilisation was at least three times higher in the heterarchical system, since in a heterarchical system information is processed in distributed processors, owing to the distributed intelligent entities.

The hierarchical control system showed to have high complexity and flexibility but low fault tolerance, whereas the heterarchical control system demonstrated to have very low complexity and very high flexibility, modifiability and fault tolerance, Fig. (2-7), adapted from Duffie.

2.8 A Hybrid Control Structure for the Durham FMS

Two types of dissimilar control structures, hierarchical and heterarchical for the Durham FMS were studied. A Hybrid control model is proposed for the control of the Durham FMS. Such a control system combines the best points offered by both hierarchical and heterarchical control structures and it distributes both decision making and control. Recently, in parallel to this research NBS [Jones 1990] has made a proposal for a hybrid control system for the shop floor control in CIM.

To exhibit the hierarchical nature of the control system for the Durham FMS, the entire system is logically divided into three levels Fig. (2-8). The master level is the highest level in the hierarchy. Part designs, system definition, scheduling and sequencing of jobs are carried out at this level. Commands and manufacturing messages are issued to the lower level where they are then translated to machine control commands. The cell level is the middle level, it contains the controller for various cells. Each cell controller is responsible for the activities of machines within its own cell. Commands issued by the master level are translated here and the appropriate commands are then issued to the machine at the lower level. Cell controllers can monitor activities within the cells and commands issued by them will be based on these sensory feedback signals.

The lowest level represents the equipment level. Equipment such as an NC machine tool which possesses its own controller, requires the ability to translate commands from the cell level in order that it may be integrated into the system. This facility allows each cell to receive part program information from the CAD system and to be able to translate commands from cell level. This translation facility requires an additional software module which resides in the memory of each cell controller.

The heterarchical behaviour of the control system is effected by the direct communication between one cell controller and another. Cell controllers can request information directly from each other about the status and progress of part production. All the system communication takes place by a software module namely, 'CellTalk'. This module utilises Common Address Memory in geographically distributed Processors (CAMP) to perform communication, described in chapter five. Cell controllers could exercise their heterarchical behaviour by taking part in a negotiation process with one another for part production. This method of part production by 'bidding', could commence once each cell had put a bid in order to claim the job, and the one with the best offer (e.g. earliest finishing time) would then win the claim [Shaw 1988].

One of the important features of the control architecture is that it can be used with a variety of manufacturers' CNC machines. If a particular machine has restricted memory capability the control function is capable of allowing the cell controller to 'drip feed' the part program details into the CNC machine's controller as the part is being manufactured. Cell controllers process parts until the manufacturing order is satisfied. Should a cell manufacture a defective part, the alarm is raised, the exact status of operation is registered before running diagnostics and then another raw part is taken from the warehouse in order to meet requirements for the production of good number of parts. Both the master and the cell level obtain their information from a common manufacturing database (dynamic and static) which is explained in chapter three.

The overall control system allows the termination of a part production session. In order to terminate the operations in different cells the hierarchical behaviour is exercised. This is done by the entity at the master level (scheduler) issuing control commands to all the cells to terminate their part production at the earliest time it is safe to do so. This shows that the master can exercise its direct control over the cells and programmable devices whenever it deems it beneficial to the system as a whole.

Once the overall control structure was designed and implemented on the experimental FMS rig the full flexibility and capability of the hybrid control model could not have been demonstrated unless the current programmable devices within the rig were modified so that control commands could be sent to them remotely without the intervention of a human operator. Each programmable device was interfaced to a cell controller where the appropriate control command received from the higher level were decomposed before issuing machine level instructions to the controller of the programmable devices. This required the design and building of interfacing modules which is described in the following section.

2.9 Remote Control of Programmable Manufacturing devices

Programmable devices in a FMS are expected to function both in manual and remote mode. Here, remotely controlled devices refers to those devices which are capable of receiving control commands from the outside world without the intervention of a human operator. Remote operation is particularly needed if a programmable device is to be integrated in a FMS environment. Unfortunately, none of the devices used in the Durham FMS exhibited such ability, therefore an interfacing module had to designed and implemented.

Although the programmable devices (CNC machines) were equipped with a RS 232 serial link, they were unable to accept control commands such as LOAD, START, STOP etc. Knowing that the upgrading of the machines would be very high in cost (half the prices of a new machine of the same type), interfacing modules were designed to mimic the keyboard operation (function keys) of the CNC machines. In particular, the interfacing modules were implemented on a CNC miller namely, TERCO-1000 manufactured by the TERCO AB of Sweden.

The microcomputers available for use for this research project were BBC microcomputers, and for a BBC to adopt the role of a cell controller, in its standard form, possessing only a limited input/output capability, it could not satisfy the demands. Therefore, an expansion board had to be designed to enhance the input/output capability of the BBCs.

The hardware of the interfacing module has two separate modules. One module provides a multipurpose input/output facility which enables a BBC microcomputer to send or receive digital signals to or from any other devices. It resides in a separate isolated box with its own power supply and connectors. This module takes advantage of two Versatile Interface Adaptors (VIA 6522), each providing two 8-bit bidirectional ports. Since the BBC microcomputer has an 8-bit external data bus, the expansion board was designed in such a way as to enable it to receive or transmit up to 16-bit in parallel, Appendix A.1.

The other module is more application specific. It may reside in the controller of the programmable device, and provide an interface with the outside world. The idea behind this module is to bypass all the unnecessary protocol conversion, data formatting and communication problems by simply tapping the information in the I/O of the programmable device at its keyboard interface. To the device this appears to be an operator actually inputting the control command via its controller's keyboard which is described in Appendix A.4.

The circuit diagram for the CNC controller was studied and its keyboard encoder circuit was identified. The circuit tracks where the keyboard encoder is routed to the CNC's Peripheral Interface Adaptor (PIA) were tapped off. A manual switch was installed on the CNC's controller to provide the toggling between the manual and remote operation. When the CNC is in remote control mode, control information can be sent to the tapped off channel, enabling the device to operate without human operator intervention. Appendix A explains all the necessary features of the interfacing module, design and its implementation.

2.10 Conclusion

Different control strategies for the overall control of the Durham FMS were described. These are listed under centralised, decentralised, hierarchical and heterarchical control structures. The advantages and disadvantages of each were highlighted. Also, two hierarchical structures for the control of automated manufacturing systems have been described. These two structures were the Advanced Factory Management and Control System (AFMCS) of Computer Aided Manufacturing International (CAM-I) and that of the Automated Manufacturing Research Facility (AMRF) of the National Bureau of Standards (NBS).

A hybrid control architecture for the Durham FMS which utilises the best points of hierarchical and heterarchical control structures was proposed. The hybrid control structure was implemented on the Durham FMS rig where it was demonstrated to function successfully in real time. The hierarchical behaviour of the system is shown by division of the entire system into three levels of master, cell and equipment. Each level receives the requirement from the level above and the feedback signal from the level below before issuing control commands to the next level below. Entities at the master level have an overall view of the system and its performance and where possible decisions are dictated to the cell controller/s for the benefit of the system as whole.

The heterarchical behaviour is demonstrated by cell controllers exchanging information with one another. Control decisions are made based on information obtained from other controllers via information exchange. Hence, control is distributed by the centre of decision making being passed from one controller to another. If a cell breaks down it will avoid having an immediate impact on the other cells by simply dissociating itself from the rest of the system. Cell controllers do not rely totally on the information exchange among themselves for job processing but receive some information on what job to process next from the master level. This will reduce the volume of data traffic in the network since the cell controllers need not talk to one another to obtain such information.

Once a cell controller had received a job name from the scheduler at the master level it would not need further instruction on how to manufacture the part and such a decision is made by the cell controller itself. This showed that the cell controller had some autonomy on the production of parts in the FMS. Interfacing modules had to be designed and implemented so that the controllers of different programmable devices were able to receive control commands from higher levels. These modules enhanced the operation of the FMS making the intervention of a human operator redundant.

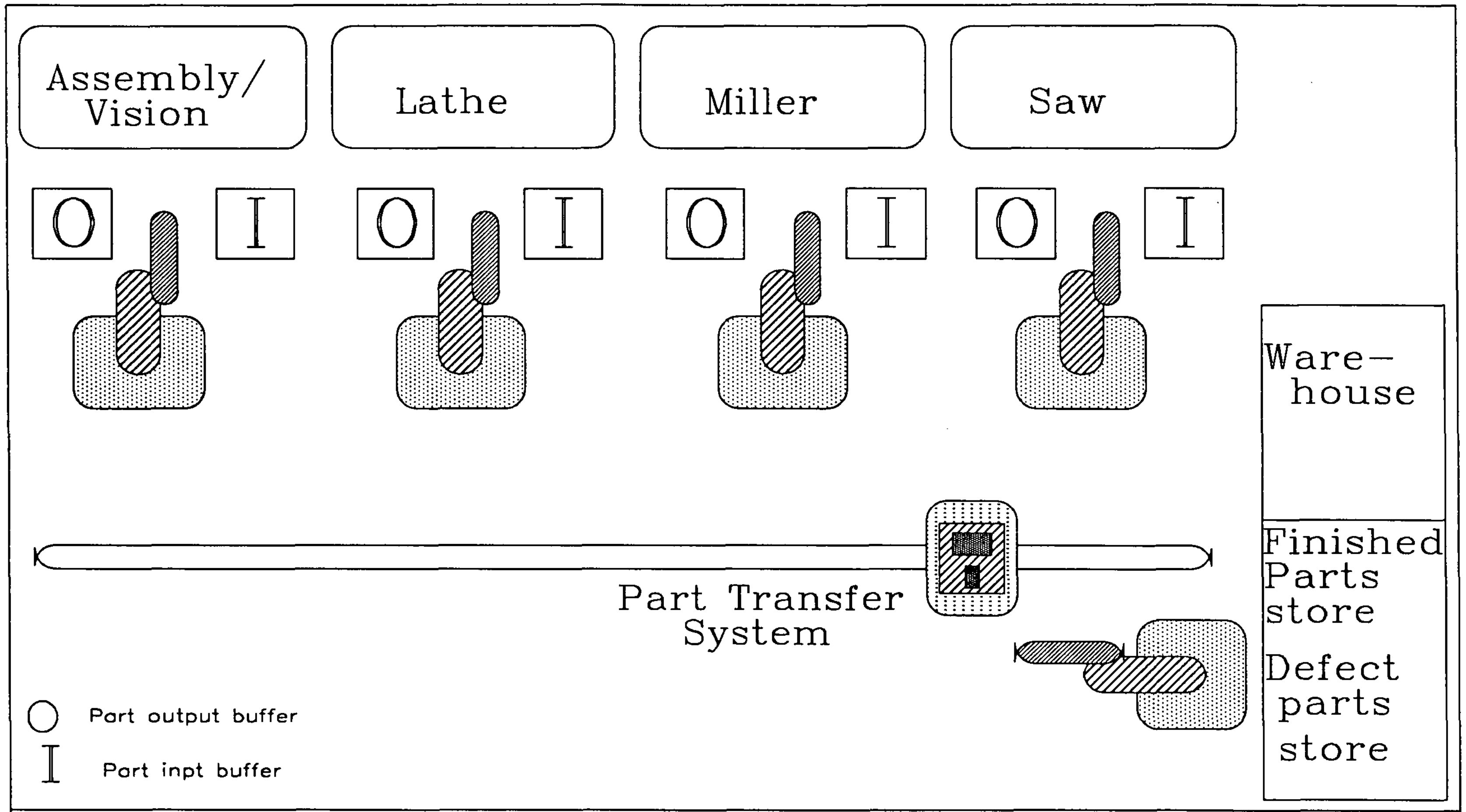
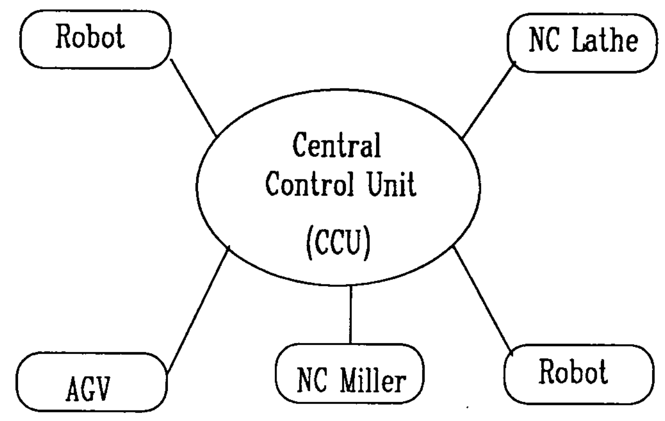
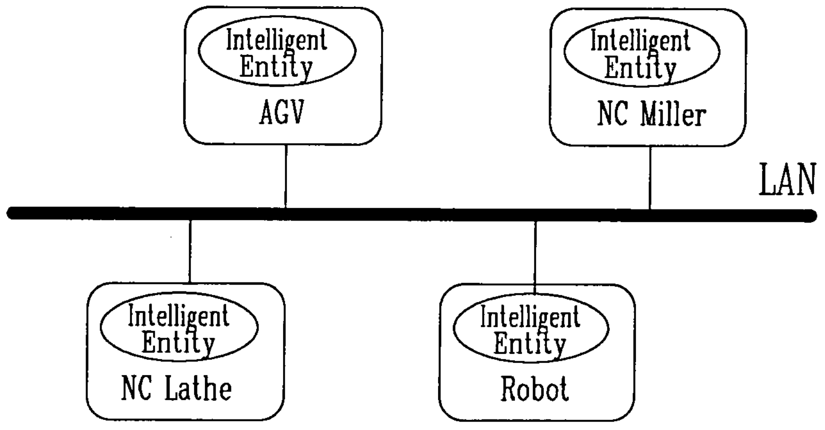


Figure (2-1) The experimental FMS set up at Durham university.



(a)



(b)

Figure (2-2) FMS with (a) Centrlised control architecture
(b) Distributed contol architecture.

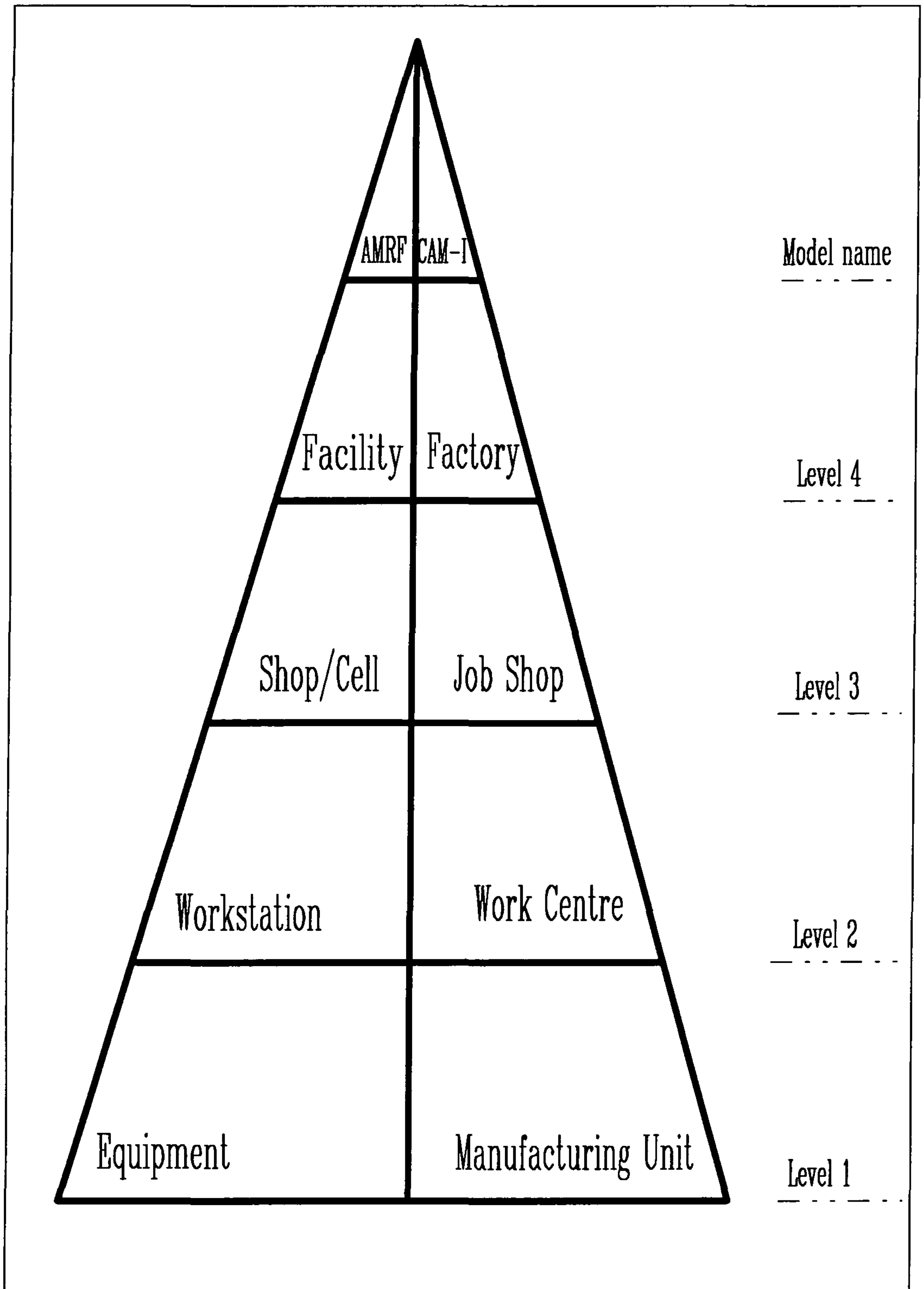


Figure (2-3) The hierarchical control architecture for CAM-I and AMRF models.

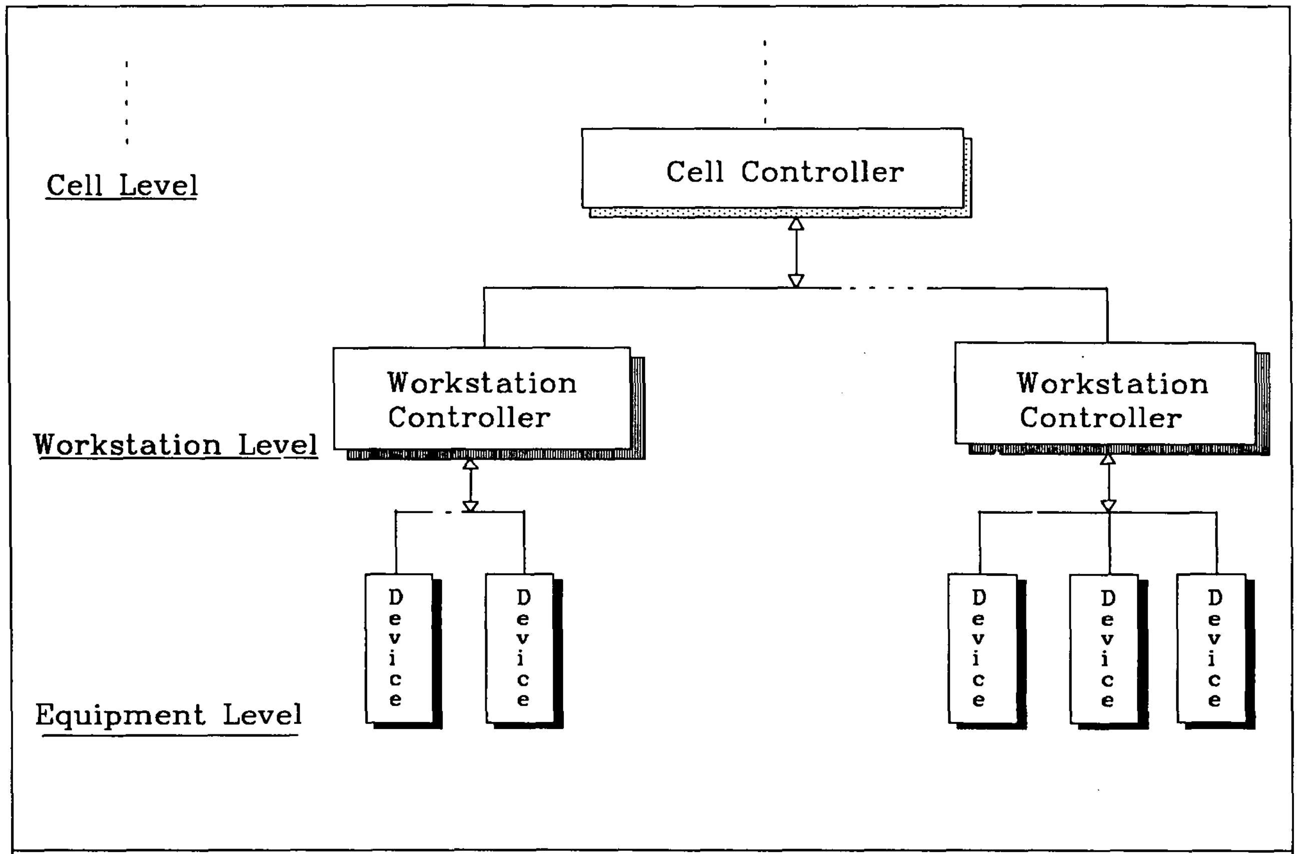


Figure (2–4) A system with hierarchical control structure.

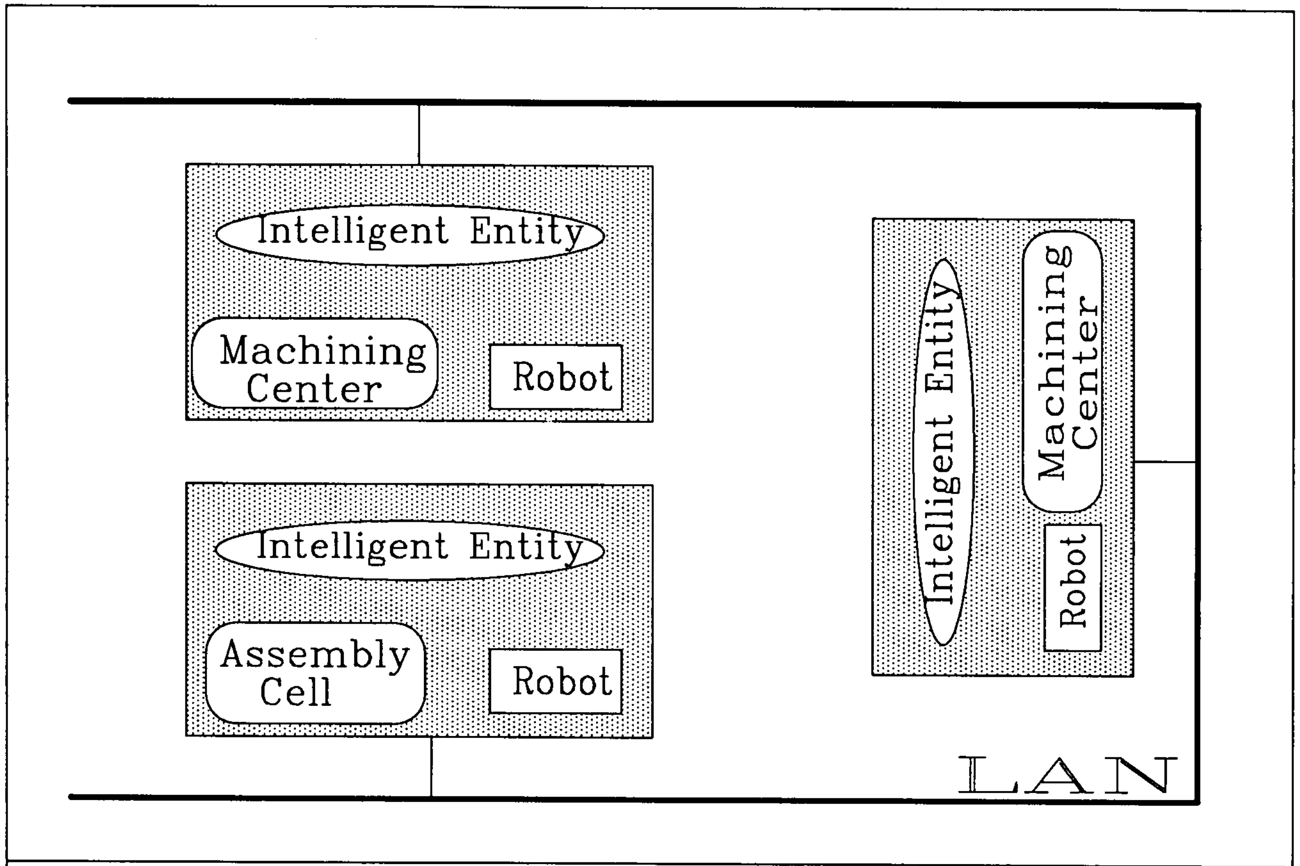
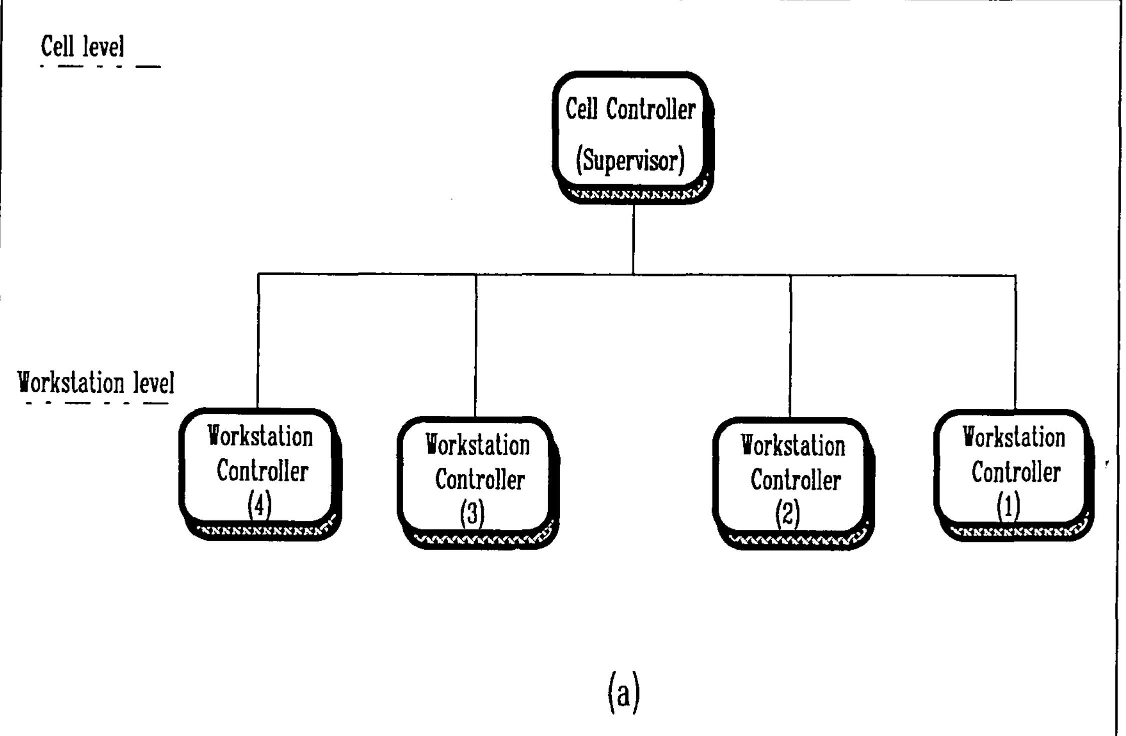
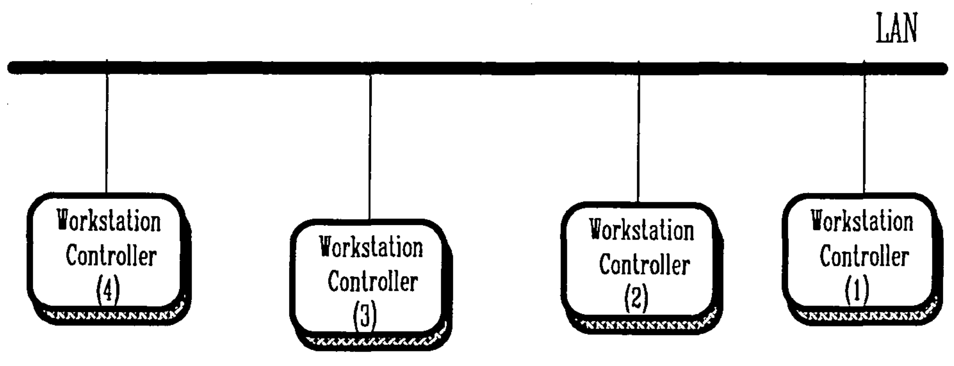


Figure (2-5) A system with heterarchical control structure.


Cell level
Cell Controller (Supervisor)
Workstation level
Workstation Controller (4)
Workstation Controller (3)
Workstation Controller (2)
Workstation Controller (1)

(a)


LAN
Workstation Controller (4)
Workstation Controller (3)
Workstation Controller (2)
Workstation Controller (1)

(b)

Figure (2-6) (a) Hierarchically controlled FMS (b) Heterarchically controlled FMS.

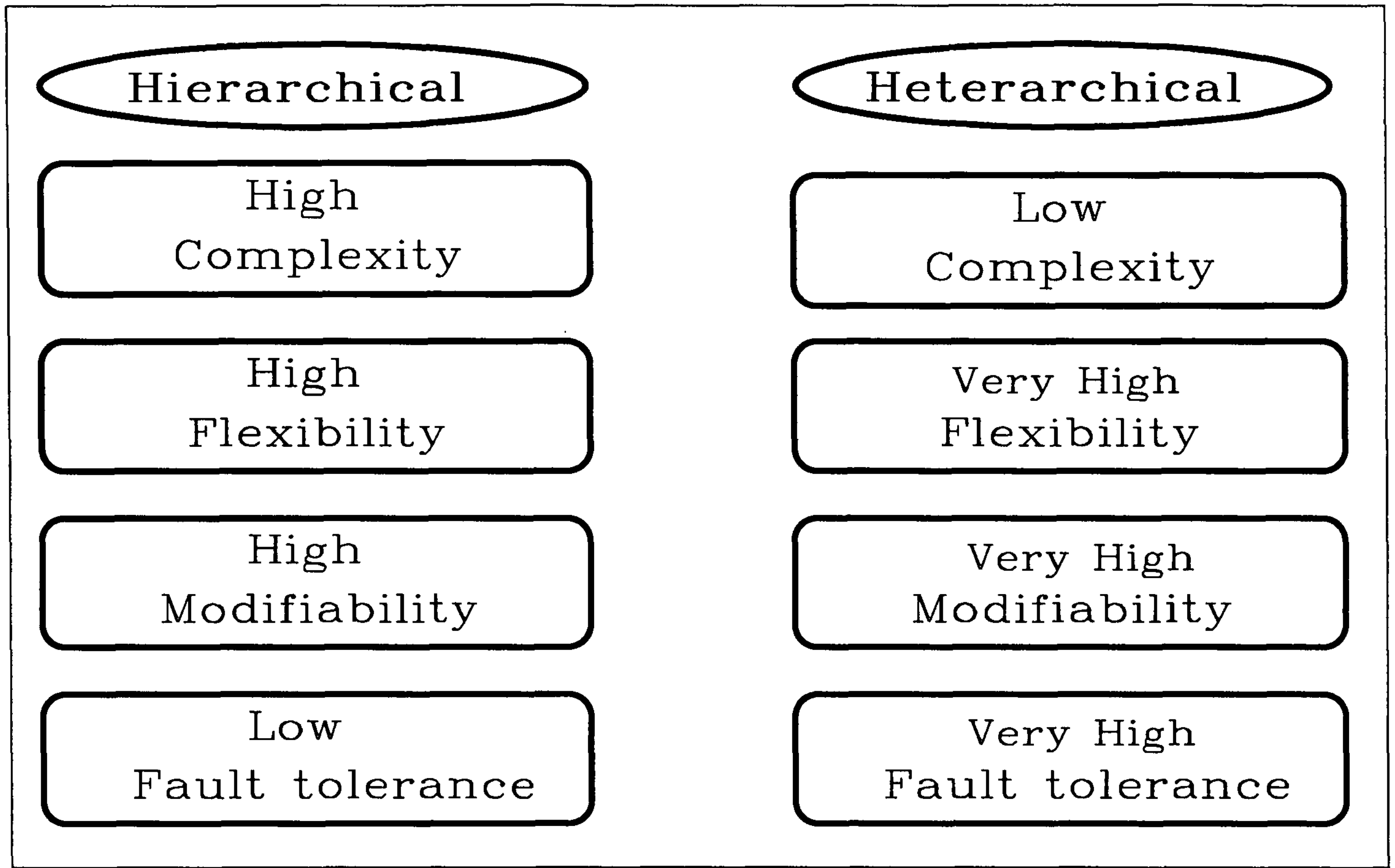


Figure (2–7) Comparison between hierarchical and heterarchical control structure.

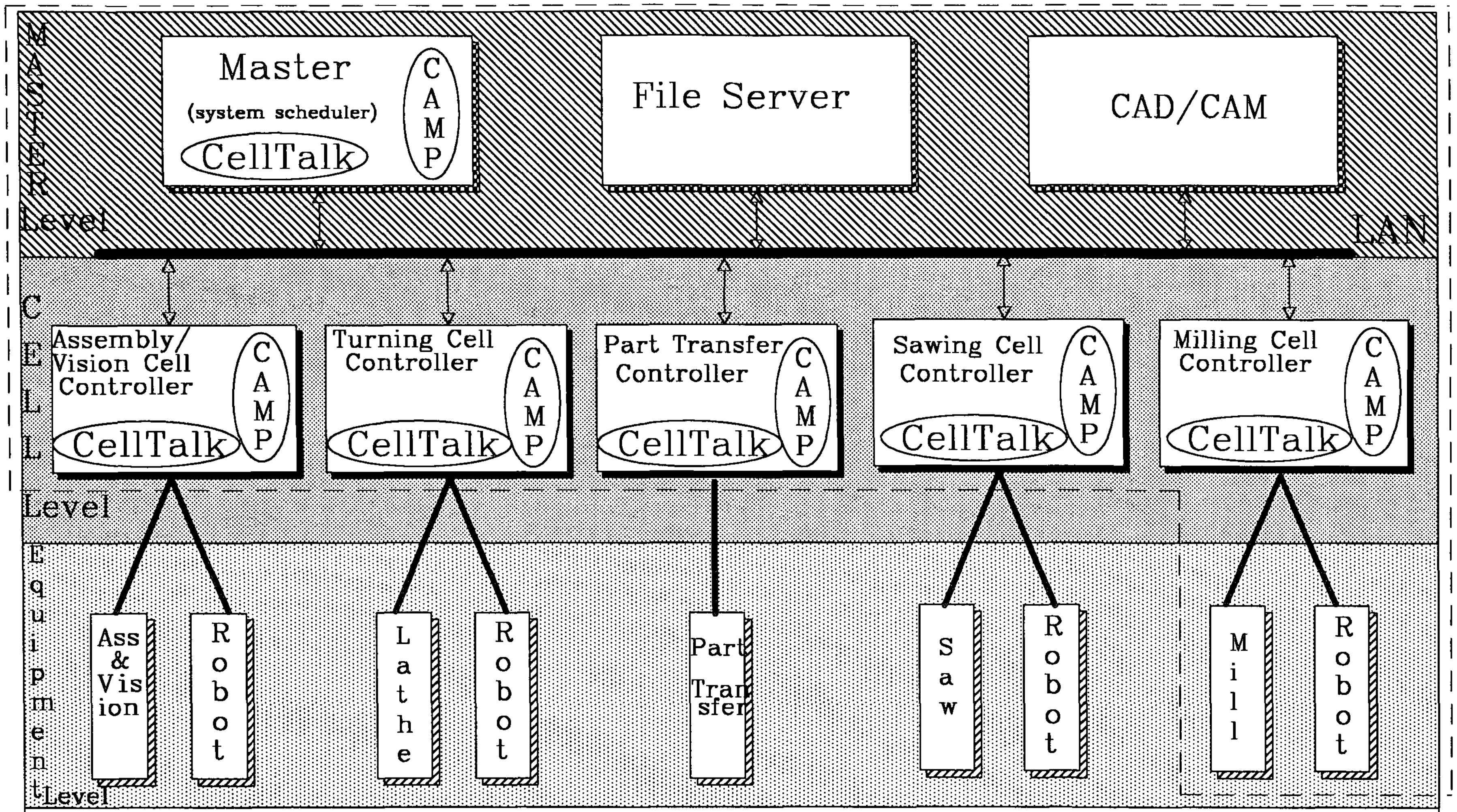


Figure (2–8) Control organisation in Durham FMS.
Broken line indicates boundary of the hardware implementation; activities in other cells were emulated for demonstration.

CHAPTER THREE
Computer Based Information System

3.1 Introduction

Each system relies on certain basic information. In Flexible Manufacturing Systems this information is needed for the part production planning, control and also managerial decision making. The volume of this information is very large and the manual handling of it will prove the system deficiency. Flexible Manufacturing Systems require a technique or system which can provide rapid and accurate access to the desired information. Currently, a manufacturing database provides such services. Information within the database is updated regularly so that high quality data may be obtained by the system modules.

This chapter explains the techniques used for information organization. Section 3.2 gives an overall view of a database system which is then followed by the manufacturing database designed for Durham FMS in section 3.3. The manufacturing database is comprised of two modules, one holding the system static data and the other keeping the system dynamic information. These are described in sections 3.4 and 3.5 respectively. Section 3.6 describes a more application specific representation of some information. This is the presentation of a part's geometry during a design period. Finally, section 3.7 explains how the part geometry is encoded in the form which can be utilised directly by a CNC machine as input data.

3.2 Information Organisation in a FMS

In a Flexible Manufacturing System the value of information is determined by the way that it is organised and used [Shaw 1988], [Hutchinson 1987]. Information may add value to an FMS if it is applied in support of a decision (i.e. it is being used to control a process). Decisions that cannot be made because of insufficient data are often a common source of problem in manufacturing environments. Alternatively, having too much data without it being organised in some way can also cause difficulties.

A database can be used to organise and store information in an efficient way [Managaki 1988]. The data is shared among different applications, but a common and controlled approach is used for adding, modifying or retrieving data. The application programs which access the data do not need to be aware of the detailed storage structure. In this way the data can be used as the basis of an information system.

A database may be defined as a collection of stored data organised such that all users' (application program) data requirements can be satisfied. This contrasts with traditional systems where the data required by an application program is bound to that application program, and the data structure is declared within the application. A database is often used as a means of interfacing various modules of a system together [Nemes 1987].

The two main considerations when implementing a database for a FMS are its structure and contents. The way in which individual items of data are internally stored in the computer defines the structure of the database. This in turn determines the ease with which various application programs can interface with the database in order to modify, retrieve or add data. Since the database is to be utilised in a manufacturing environment, its content must include information for the successful part production. This may be a digital representation of the geometry and technology of the part designs, which is stored to a high precision.

For a database to be able to store, process, transmit and display graphical data as well as the alphanumeric data, a distributed approach may have to be taken [Jablonski 1988]. This is because of the sheer size of the database and, the changing nature of software practices will make utilisation of a single database impractical. A distributed database assures data structure compatibility among independently developed application programs [Berra 1990]. Each application may request data or supply data to a distributed base regardless of its brand or model. The data may be stored on a single shared computer (i.e. a file server) or different portions of it may be stored on different computers throughout the network.

As the expectation of efficient part production at reduced cost by the FMS increases, the need for efficient management of data in the FMS becomes more apparent. A Database Management System (DBMS) can provide the means for supporting and managing of any number of independent databases. Its function is to provide an interface between the application programs and the database. DBMS will ensure that application programs are no longer sensitive to changes in the structure of the database, unlike traditional file-based systems, where a change in the structure of data requires change in the application program.

A DBMS should allow data to be easily and quickly retrieved, added, searched, copied or deleted. It should also support a data security mechanism where it limits the access of unauthorised users [Hughes 1988].

Database Systems are generally based on one of four data models, namely: Hierarchical, Network, Multi-user distributed and Relational data models [Ranky 1986]. Relational database gains its popularity among FMS and CAD/CAM systems due to its flexibility over other databases and relative ease of design and modification of its data structure [Upchurch 1988]. The relational database is structured from files consisting of records, and records of files. Files are linked by the fields of records, allowing for the establishment of new links between files at any time during the lifetime of the database.

The areas involved in a Flexible Manufacturing System include design, process planning, machining, inspection and assembly. In order to ensure efficient utilisation of such data a manufacturing database may be incorporated.

3.3 Manufacturing Database

A typical manufacturing database contains several major sources of information. These include the part design , routing, operation sequencing and tooling information. The part design information contains a detailed description of the part design and type of operations to be performed on it. During the process of part production some cutting tools may be more scarce than others and in order to keep the cost of tooling low, information on utilisation of alternative tools may be stored. This information gives the system greater flexibility, since the production of a part will no longer be restricted on the availability of a particular tool. Routing information defines the route or possible alternative routes that a part can take during the process of part production. This information will play a major role in the utilisation of machines and efficiency of the system to its best. Operation sequencing information provides data on the actual sequence of operations which a part has to undergo. Here the sequencing referred to is the sequence of operation within a manufacturing cell, or sometimes better known as intra-cell sequencing.

Tooling information provides data on the tools which are available. This data may include the tool number, its type, length, life, material, cost, status, accumulated usage time and its supplier. For a manufacturing company which is engaged in metal cutting, this information may be vital [Hollingum 1989]. Designing a database to include all this information requires both a powerful software package and the hardware on which the database could function fast and reliably. The hardware in the Durham FMS imposes restrictions in the design and utilisation of those kind of database packages, therefore a separate database module was designed to be suitable for the existing setup.

A manufacturing database has been designed and developed for the Durham FMS to ensure efficient utilisation of common system information. There exists an interface between the application programs and the database, which consists of a general set of routines to allow the user to perform functions such as deleting, adding, modifying data, etc., from the database. Data files are made of records and the records of field. Each data file comprises a header record followed by the records. The software provides the user with a tool similar to a simple Structured Query Language (SQL). Application programs residing on distributed processors are all allowed simultaneous read access to a datafile, however, only one application at a time is allowed to write into a record. A number of the SIT modules utilise the database to satisfy their needs.

The information within the manufacturing database is divided into two parts, "static" and "dynamic" [Ketcham 1988]. The static information refers to the information which is comparatively unchanged and gives information about part production specifications. The dynamic information relates to the information which changes as the status of the production system changes. Such manufacturing database has also been proposed by AMRF of NBS [Simpson 1982].

3.4 System Static Data

In AMRF model the system static data contains part dimension data, desired grip points for robot handling, material and tooling requirements, process plans for routing and scheduling , cutter location data files needed for various machining operations, and data related to feeds and speed for programmable devices. Such databases contained high volumes of detailed part and system information for management, control and production of parts in a manufacturing environment. However the hardware in the Durham FMS could not cope if all this information was included and therefore the system static data was defined in a different way which is explained in the following paragraphs.

In the Durham FMS the system static data module resides at the master level and allows the user to define the manufacturing environment. The output from this module is written into a file called the System Definition (SD) file. System information such as cell type, cell controller address for communication over the LAN, name of cell controller software, cell condition, status and cutter location data file names are entered at this level. The content of the SD file is referred to as static data since it is not permitted to change once the process of part production begins.

The experimental FMS comprises four different physical cells, and one record is allocated to each cell to hold all the relevant cell data. The size of these records is arbitrary and can be set differently at the commencement of system definition if necessary. This will give the flexibility to define the systems with various complexity, and in the case of expanding the existing set up, it will allow unlimited system parameters to be included.

One System Definition (SD) file is created per job and once a SD file has been created it cannot be changed, only the contents of associated static data files may be modified. Within the Durham FMS a job is defined as a product to be designed and manufactured through single or multiple stages. At each stage one or a number of operations may be performed. The design process is carried out in the CAD module (described in 3.5) and the process of part production takes place in one or a number of manufacturing cells depending on the description of part design. Each job may have a different batch size according to the requirements.

All the software modules within SIT are developed in such a way as to provide the user with menus, making it as user friendly as possible. There exists a main program namely "MASTER", Appendix F, which is loaded into a computer (i.e. master computer at the master level) at the start of system initialisation. Executing this program offers the user choices from the menu. Each option on the menu may load and execute other programs, and it always returns to a menu when the execution of the current task has been completed.

New jobs can be created by choosing option No. 1 of the MASTER program. The user is then entered into the manufacturing database routine of "MAN-DBASE" program, Appendix F, where it is provided with a $ prompt and a second menu showing a list of commands which are allowed to be used. Commands are only accepted if the correct syntax is entered. For example, to create a new job with a JOB7 job name, the following must be entered after the prompt; CREATE JOB7. User may enter only the first character of the command (i.e. C JOB7). The command list is made to be self explanatory and the list of commands can appear on the screen every time character M is entered (for menu) at the prompt.

A detailed description of the database can be found in Appendix B.1.

In order to simplify the process of job definition, the user is allowed to choose for the new job to be similar to an existing one. Then by editing only the record or records of the created job, the new job can be defined. Since the system static data of each job is held in records of SD file, and if one record is used per production cell to contain the information related to that cell, jobs can have up to n number of cells.

Each physical cell may be defined logically more than once with a different name but the same network address. This feature of SIT enables non-flexible NC machines with small memories to be utilised in a more powerful way to accept large part programs in the 'Drip-feed' mode.

Defining a new job will involve first to create a SD file and then the contents of records within the SD may be filled. The SD file consists of a header record which identifies the information about the fields of the records, containing system static data. This information is about the number of fields used in each record, the name of the fields, their type (numeric or character), width and number of decimal places (if numeric) and finally the number of records contained in the datafile. Figure (3-1) shows the structure of system static data. One record is allocated per each production cell (physical or logical), and each record contains eleven fields. Each field holds a specific type of information which is used in the process of part production and are described as following.

Operation type (OP.TYPE) is held in field one to identify the nature of the operation within the cell, whether is milling, assembly etc. Each cell posses a cell controller program which controls the activities within that cell and is specific to that cell, this information is held in field two as cell controller name (CC.NAME).

The condition of a cell (CC.COND) is defined in field three, whether the production cell is active or inactive.

All the controlling computers (e.g. cell controllers at the cell level) are connected to the LAN. Each computer has an unique address on the network (i.e. station id) which may be addressed by others. This address (STAT.NO) is defined in field four. Fields number five and six hold information on the number of part program (NOF.PP) to be used in the cell and the name of part program (PP.NAME) or part programs.

During a typical period of part production a part (parts) has to visit one or several manufacturing cells, depending on the description of part design. The order in which a part is allowed to visit different cells is very important. Every manufacturing cell will have a sequence (sequences) number which designates its position to accept a part in the process of part production. For example milling and turning cells with sequence numbers of 3 and 4 respectively, means that a turning operation on the part cannot be performed until the milling operation has been carried out on the part. This sequence number (SEQ.NO) is held in field seven.

The average setup time (SETUP.T), which is defined as the time spent for work (part) mounting or pallet changing, is held in field eight. The downtime (DOWN.T), that is the total time that a manufacturing cell is not doing productive work, is contained in field nine. The time taken for a part to be processed (PROCES.T in a manufacturing cell is held in field ten. Finally field eleven holds the defect rate of the cell. Information such as setup time, downtime, processing time and defect rate of cells are utilised in the FMS cell requirement check modules which are discussed in chapter four.

Each field holds a certain type of information about the FMS, and all the programs utilising the manufacturing database rely on this information. For the database to find a certain type of information on FMS it has to search through the fields of records. For example to obtain the STAT.NO for a particular cell, it first searches for the record in which the data is held about that cell, and knowing that the relevant data is held in field number four, it then obtains the data from that field.

However the user may wish to define the fields in a different order. This has been made possible by the database utilising a numeric variable namely "keyX" to identify what fields contain which type of information. This information is preset at the beginning of the database program (e.g. 'STAT.NO=4', 'key4=STAT.NO' for station number, and whenever the manufacturing database requires data on the station number it uses numeric variable key4). Should the user change the order in which the data fields are defined, then the value assigned to the keyX has to be adjusted accordingly at the start of the database program. For example if the order of data field is changed in such a way that the data is to be held in field six, to be the station number then the following change must be made; 'STAT.NO=6'. In this way the routine within the database will require no change since the numeric variable key4 is also set automatically to give information about the station number.

System static data is used to define jobs and resources, and to initialise and start a process of part production, however, the overall control of the FMS and control of manufacturing cells heavily relies on the system dynamic data.

3.5 System Dynamic Data

Since all the controllers of manufacturing cells have access to the manufacturing database, real-time control data may be accessed from or written to the database. System dynamic data are those data with frequent changes, such as programmable machine status, part location, resources status etc. The system static data may be prepared off-line but all dynamic data is to be gathered on-line in real-time.

In a typical system with a dynamic database cell controllers will monitor the status of devices, process, parts etc. within the cell and this information is then stored onto appropriate records of files in the dynamic database. All the decisions made by the cell controllers are based on the feedback information held in the dynamic database after being checked against predefined rules in a state table [Drolet 1989]. If the requirement is satisfied, the cell controller will issue the next command from its list to the programmable device; otherwise it will compare this state with those of states defined for errors and malfunctions. Should it find an error or malfunction, it checks through the error recovery routine for the salvage of equipment operation or possible total termination of the operation.

Since each cell controller, after a monitoring cycle, writes the state of the monitored signal into the dynamic database , the cells can pass information to one another via the database. Therefore the database acts as a common memory and it makes the system modular and as long as an entity (i.e. a programmable device controller) can read from or write to the database it can be added or eliminated from the system with the minimum of impact on the other components of the system. In this way the database (static and dynamic) can provide the means for integrating resources supplied from different vendors.

However if the dynamic database is implemented in the aforementioned way, each cell controller requires the use of the LAN for both writing and reading various status to or from the database, since it resides in the file server or any other type of data storage system and access to it can only be made via the network communication. As a result the volume of traffic on the network could rise severely and if the LAN does not have the deterministic feature (i.e. token rig or token bus, explained in chapter five) control functions may suffer in their decision making procedures. For this reason a dynamic database which utilises Common Address Memory in distributed Processors (CAMP) was designed and implemented for Durham FMS.

An area of memory with the same base address is reserved in all cell controllers so that the cell's dynamic data can be held in a distributed manner. In this way each cell controller has its own dynamic database to which it writes various cell states locally. Should a cell require information on other cells' status, it uses the communication protocol to gain direct access to the dynamic database of the desired cell before performing a read operation. Since the system's dynamic data storage does not reside in a database (i.e. in a file server) but rather in the memory of the cell controllers, data transfer over the LAN is not needed for a write cycle. Hence the volume of network traffic is halved. Another advantage of dynamic database is that reading from or writing to a memory location is far faster than that of a disc drive where a dynamic database could have been held.

The base address of the memory in BBC which has been set aside for system dynamic data is 7B00 (hex). There are at least two hundred bytes beyond the base address which can be used to store cell dynamic data. The same base address is used for all controllers (cells and master) and a name is given to each memory location above the base address. For example, the name "`in-partbuf`" is given to memory location 7B06 (hex) and the content of this memory location is representative of the status (number of parts) of input part buffer in that cell. The name "`in-partbuf`" will have the same address and meaning in all cell controllers. In this way all the reserved memory locations with a common address will have the same name and their contents will have the same meaning throughout the controllers (cells or master). Table (3-1) gives a list of reserved memory locations together with their contents.

The content of reserved memory locations in the cell controllers are set by different entities. For example, the memory location to represent the status of a robot within a cell is set by the robot controller. The status may be shown as "`busy`" or "`free`" by setting the contents to 00 and FF (hex) respectively. The content of each memory location may be set to 256 different states. This will give the opportunity for the definition of new states as the complexity of device increases. The dynamic database is used by several modules such as cell controller software, the scheduler and `CellTalk` which are discussed in the appropriate sections.

Since the control of part production in a FMS is largely dependent upon the information gathered from the part design and the description of how it is to be manufactured, a simple drawing package, together with the post processing facility, was developed.

3.6 Representation of Part Design Information

A simple drawing package has been developed to allow the user to design simple shapes in $2\frac{1}{2}$ D, that is, having full freedom in two axes (X and Y direction in one horizontal plane) and limited freedom in Z (vertical plane). This module is named "CAD10", Appendix F and resides at the master level and information from it is written into the system static data.

The BBC microcomputer has eight different "modes", five of which can be used for graphics, with different screen resolutions. Depending on the particular mode, 10K or 20K of the available memory (32K) is used. Mode four was chosen since it requires 10K of memory, giving high resolution (for graphics) as well as leaving space for the program itself. The screen resolution in mode four is 320x256 with two colours.

CAD10 module takes advantage of two separate windows; text and graphic. The screen is divided vertically into two sections. The portion on the right is the text window, where a part design menu, together with the design information, is present. The portion on the left is the graphic window, where design of the part takes place. CAD10 has been developed to allow the user to design simple prismatic shapes and the design information is converted into a form which is understandable to the milling machine of Durham FMS. Appendix B.2 describes the detailed programming of CAD10.

The design process commences once the user has executed the CAD10 program and inputted the design name and the blank part dimensions. The design procedure is simplified by allowing the user to draw the shape of the part to be manufactured. Drawing of lines or curves in the graphic window can be considered as the representation of the cutting path of the CNC machine tools. The cursor (flashing dot) on the graphic screen may be moved around by using the arrow keys and, by choosing the appropriate keyword from the text menu, a simple shape can be designed.

As the utilised screen size is small, and in order to maximise possible resolution, a scale factor has been calculated. This enables a workpiece of any size to fit as closely as possible onto the screen in at least one dimension. In addition, a grid is printed onto the graphic window to ease the design process for the user. The position of the cursor is continuously updated while it is manoeuvred about the screen and is shown on the text window.

During a design process the user can send the cursor from one point to another on the screen. Depending on the letter chosen from the menu in the text window, no line, a curve or a straight line can be drawn. Depressing letter 'M' from the menu for "MOVE" will cause the machine tool's cutter to move from one position to another without cutting. This is known as a rapid feed for the machine tool by the CNC-1000 (controller for the milling machine). A curve may be drawn by sending the cursor to a position and then entering letter 'C' at the keyboard. This is the quadrant of a circle, the start and end position of which has been defined and which represents the circular interpolation of a quadrant of CNC-1000. A straight line can be drawn between two points by depressing letter 'L'. For CNC-1000 this will have the meaning of linear interpolation between the two points.

Removing of lines or curves can be performed by depressing letter 'R', as this can delete up to five previous lines and curves. The depth of cut or the feedrate may also be changed at any stage of the design by entering letter 'D' or 'F' respectively. The current setting of feedrate and depth of cut is shown on the text windows at all times.

Once the part design has been completed, depressing letter 'E' will end the session and user is given option of having a hard copy of the design. Note that the package does not build a mathematical model of the part, rather it gathers sufficient information to enable the machining process to be executed, therefore once the image has disappeared from the screen, it can not be brought back onto the screen. The data from the design stage is formatted and stored in such a way that it can be utilised directly by the CNC controller of the manufacturing cell.

3.7 Postprocessing of CAD data

Throughout the process of part design the drawing is post-processed for the CNC miller and the generated data is stored. There are two ways in which this can be carried out. One, to store the data in arrays within the available memory to the user, the other to write the data onto disc. Due to the shortage of memory within the BBC microcomputer the former way of storing data is considered wasteful of memory if any significant amount of it is used. Hence, the data was stored in the form of strings onto a file on the disk. The file handling capability of BBC allows random access of data within the data file and this proved useful for the editing purposes.

As with many other CNC machine tool controllers, CNC-1000 will accept the part design data in the form of a part-program. This is in the form of a series of data blocks that defines which control signals the controller has to send to the motors (i.e. axes motors, spindle motor, actuators) next. In each block a number of "functions" must be set to define the specific operation for the controller. These are; "N" for block number, "G" for type of interpolation (circular, linear, etc.), "F" for feedrate (mm/min), "S" for spindle speed (high or low), "T" for tool number, "M" for auxiliary (e.g. spindle rotation direction), "X Y Z" for absolute coordinate of the end of the path and "I J K" for quadrant center (for circular interpolation only).

For the CNC-1000 to be able to receive the part design data (part-program) so that it can generate the control signals to the milling machine tool to machine the part, the data should be formatted in the following way. The data must start with a '%' followed by the first block of the part-program. The first block of data must contain codes for functions such as spindle speed, tool number, spindle rotation and feedrate. Positional information is only accepted if a letter (X, Y, Z, I, J) signifying to the CNC-1000 which coordinate is being referred to, is followed by a five digit number. For example, to represent a position of 20mm in the Y-direction of the workpiece surface, the string has to be encoded as "Y02000". The last data block will reset all the functions, such as switching the spindle off and going to home position followed by a linefeed and a ":".

When data is being stored on the disc it is not required to include a block number at the start of every block, as incrementing the block can be done when the data block is being sent to the CNC-1000, by sending the code for linefeed at the end of every block.

3.8 Conclusion

An information handling system module was developed to organise the data required for the management, control and production of parts. This module is comprised of two parts, one keeping the system static data and the other holding all of the system dynamic data. It was found that the utilisation of Common Address Memory in distributed Processes (CAMP) was a very efficient way of dynamic information handling. The system allowed the users to define their own manufacturing environment. The software has the potential to be applied to any type of manufacturing environment. It provides users with the flexibility to define n number of manufacturing cells. The software also allows equipment from different vendors to be integrated into a FMS without encountering communication problems.

The system can accept part design information from any design package as long as it has been provided with a postprocess datafile of the design. However, sections 3.6 and 3.7 describe a small package developed for Durham FMS, which can be utilised by users for design and postprocessing purposes. As an example, design data was postprocessed for the milling machine of the FMS. Therefore, the subject of discussion in section 3.6 is application specific. This was chosen to demonstrate how design data can be adapted for the manufacturing purposes of a specific machine. There exist several commercial packages which can be utilised to perform design (e.g. AUTOCAD) and then to postprocess (e.g. SMARTCAM) the design information into a format suitable to specific programmable devices.

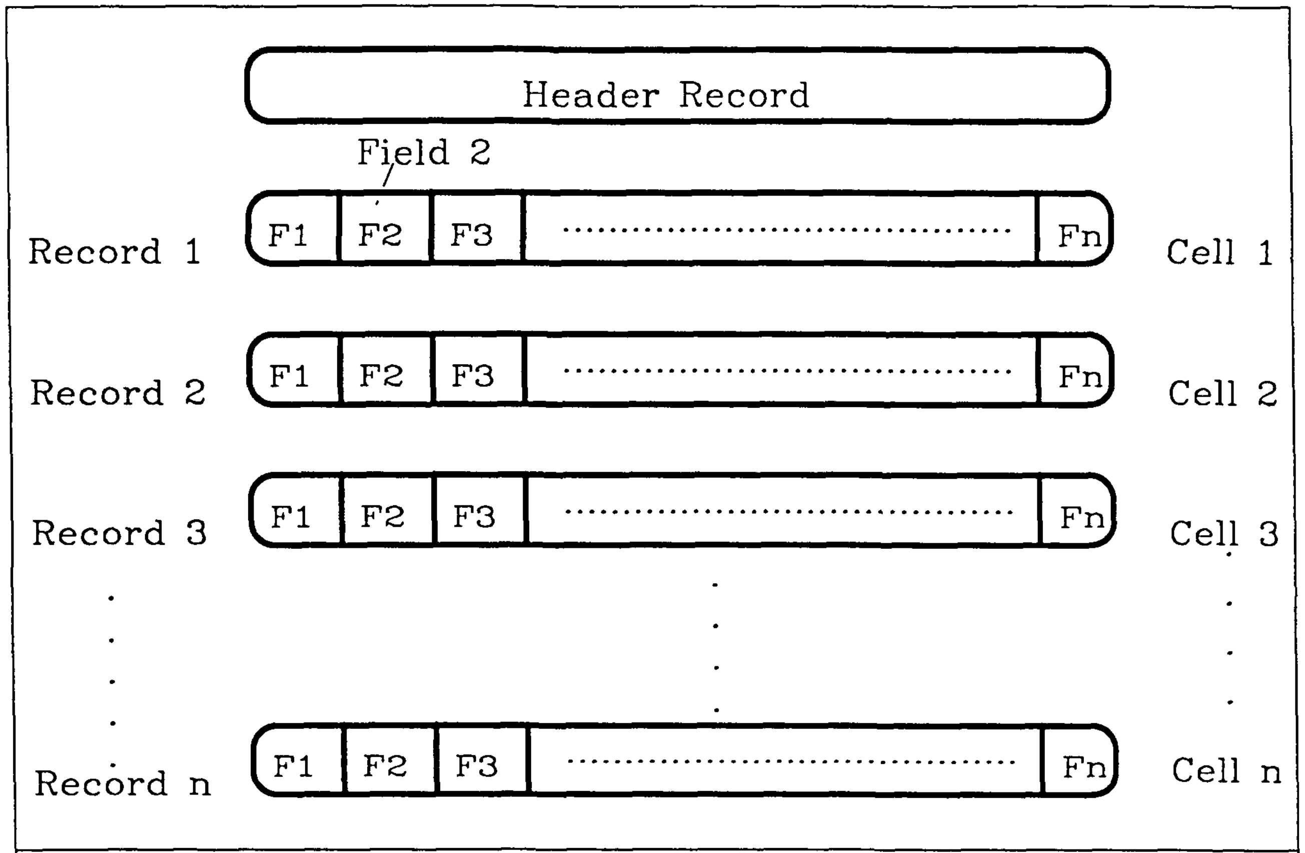


Figure (3–1) The structure of system static data.

Table (3-1) Reserved memory locations for system dynamic data storage.

Address (Hex)	*Address name*	*Set by*	*Read by*	*Memory content description (Hex)*
7B00	logi-assoc	MA	CC	00=disconnect FF=Connect
7B01	client-st-no	MA	CC	station number
7B02	client-port	MA	CC	port (communication channel no.)
7B03	server-resp-mes	CC	MA& CC	00=not-ready FF=ready
7B04	mass-stat	CC	MA& CC	00=processed FF=processing 01=message-not-recognised
7B05	prog-stat	CC	MA& CC	01=down-load successful FF=program running 00=completed
7B06	in-partbuf	CC	MA& CC	number of parts in the input buffer
7B07	out-partbuf	CC	MA& CC	number of parts in the output buffer
7B08	production-period	MA& CC	MA& CC	00=completed FF=started 09=default state
7B09	upload-prep	MA& CC	MA& CC	00=completed FF=in-progress
7B0A	part-trans	MA	CC	FF=part to be transferred to/from the cell 00=part transfer completed
7B0B	where-part-trans	MA	CC	01=part transfer from the trolley to the cell 02=part transfer from the cell to the trolley 09=default state
7B0C	server-resp-part	CC	MA& CC	00=not-ready for a part transfer FF=ready for a part transfer
7B0D	robot-stat	R	CC	00=free FF=busy
7B0E	trolley-stat	CC	MA	00=free FF=busy
7B0F	device-stat	MA	CC	00=completed 01=idle 02=stopped 03=critical operation in progress 04=device reset 05=device inoperable 06=uncorrectable error detected 07=correctable error detected 08=diagnostic running
7B10	defect-part-stat	MA& CC	MA& CC	FF=part defective 09=part is not defective
7B11	nof-defect-part	MA& CC	MA& CC	number of defective parts
7B12	total-nof-parts	CC	MA& CC	total number of parts in the cell
7B13	nof-processed-parts	MA& CC	MA& CC	number of processed parts

Key:		
	CC	: Cell Controller
	MA	: Master
	Address	: Reserved memory locations address

CHAPTER FOUR
Production Planning and Control

4.1 Introduction

Production Planning and Control (PPC) is concerned with all of the activities from acquisition of raw materials to delivery of completed products. It also deals with management problems and techniques for their solution, as well as with the linkages or interactions among particular problem areas. Production planning may be described as the process of organising material and component availability and of optimising the use of productive capacity in a manufacturing organization [Weatherall 1988]. It provides the control strategy in accordance with requirements received from the higher planning functions and feedback on the real status of the manufacturing process [Foote 1988].

The subject of this chapter focuses on production planning and the real-time control of Durham FMS. Section 4.2 gives the logical position and a brief description of each of the main building blocks of a PPC system. Having given an overview of the PPC system two of its modules are then expanded further. These are the Capacity planning and the Production Activity Control. Section 4.3 describes the developed FMS cell capacity check module, the offered services of which include utilisation as a simulation tool for checking the equipment requirements. This is followed by section 4.4 describing the building blocks of a PAC system. Some of the algorithms which are currently used in shop floor scheduling are explained in section 4.5. The functionality of the real-time scheduler developed for the Durham FMS is described in section 4.6 and finally the cell controller module which takes charge of the in-cell operations is discussed in section 4.7.

4.2 A Structure for Production Planning and Control

There exist, in any manufacturing company, three distinct activities or phases for the Production Planning and Control (PPC) system [Vollmann 1988]. The Production Planning and Control structure described here is a very brief overview of a PPC system and does include some of the elements which are used in a real manufacturing environment. The reason behind this is to give an introduction and to establish the logical position of two of the modules which have been designed and developed for the Durham FMS. These modules are the FMS capacity planning and a scheduler for the FMS activities which are described in the subsequent sections to follow.

Establishment of the overall direction for the company is the first activity of a PPC system. This phase establishes the company objectives for production planning and control. The management plan is stated here in manufacturing terms, such as product options or end items. The manufacturing plan must be consistent with the company's direction and the plans for other departments of the company. A game plan that links and coordinates the various departments of the company provides the overall direction. The responsibility for this game plan is of top management. The second PPC system activity is the detailed planning of material flows and capacity to support the overall plans. The third and final PPC system activity is the execution of these plans in terms of detailed shop floor scheduling.

Figure (4-1) shows a simplified structure consisting of four levels for Production Planning and Control (PPC) system. Level four represents the top management which is responsible for managerial objectives to develop an integrated game plan for which the manufacturing portion is the production plan. The game plan reflects the strategy (e.g. increased market share) and tactics (e.g. increased inventory for improved service) that used by the company. The manufacturing part of the game plan is the production plan and is illustrated in the third level.

The production planning problems are generally divided as aggregate and disaggregate planning. In aggregate planning a single variable representing the total production of all products using some natural unit such as volume, weight, units, machine-hours etc., is used as a measure of aggregate output. Disaggregate planning is concerned with production plans for each product and is better known as Master Production Scheduling (MPS) which is explained later.

The production planning module as shown in the third level of Figure (4-1) provides a direct and consistent dialogue between manufacturing and top management. The production plan is not a forecast of demand. It is the planned production, stated on an aggregate basis, for which manufacturing management is to be held responsible. The production plan for manufacturing is a result of the production planning process. Inputs to the process include sales forecasts; but these need to be stated on the basis of shipments, not bookings. This is necessary in order for the inventory projections to match physical inventories and in order that the demands on manufacturing are expressed correctly with respect to time.

The demand planning also exists in the third level of the PPC system and it encompasses forecasting customer/end product demand, order promising, order entry, etc. Basically, demand planning coordinates all activities of the business that place demands on manufacturing capacity.

Master Production Schedule (MPS) is the disaggregate version of the production plan. MPS is a listing of what end products are to be produced, how many of each product are to be produced and when to be delivered [Groover 1984]. The

MPS takes into account capacity limitations, as well as desires to utilise capacity fully. This means that some items may be built before they are needed for sale, and other items may not be built even though the marketplace could consume them. The master production schedule must be based on an accurate estimate of demand and a realistic assessment of its production capacity. The MPS serves as an input to the Material Requirement Planning MRP module and resides at level two of the PPC system.

The MRP determines (explodes) the period-by-period plans for all component parts and raw materials required to produce all products in MPS [Westerdale 1988]. This material plan can thereafter be utilised in the capacity planning system to compute machine center capacity (and/or labour) required to manufacture all the component parts. MRP acts as a translator of the overall plans for production into the detailed individual steps necessary to accomplish those plans and it provides information for capacity plans.

The development of capacity plans is a critical activity which coincides with the development of the material plans. The benefits of an otherwise effective production planning and control system cannot be fully realised without the provision of adequate capacity or recognition of the existence of excess capacity. Capacity requirements planning means the comparison of the available capacity with the required capacity. Productive capacity is measured in units, and it refers either to the maximum output rate for products or services or manufacturing resources available in each operating period, i.e. shift [Ranky 1986].

Finally, the Production Activity Control (PAC) having the role of the execution system, exists at the level one of the PPC system. PAC is sometimes referred to as the gateway between the execution layer on the shop floor and the remainder of the manufacturing and production management system. PAC establishes priorities for all shop floor orders at each work centre (or cell) so that the orders can be properly scheduled [Browne 1988]. It also deals with the management of the detailed flow of materials on the shop floor. An effective PAC system can lead to due date performance that ensures meeting the company's customer service goals and also reductions in Work-In-Progress (WIP) inventories and in production lead-times.

Having established an overview of a PPC system, the following describes the FMS capacity planing module for which a mathematical model was developed.

4.3 FMS Capacity Planning

Capacity planning is a crucial decision since it affects both short and long term planning and control of a manufacturing organisation. Although capacity planning in general can be utilised in many levels of PPC system, here it is attempted to show its effect on the manufacturing side of the FMS. Capacity planning can be applied to the planning process to decide whether the existing capacity is large enough to take on a newly arrived order and also for controlling capacity in the existing manufacturing system.

A mathematical model was used to calculate a measure of capacity that would enable FMS cell and system requirements to be identified for each batch for a selected production route in the FMS [Ranky 1986]. The program can be used as a simulation tool for FMS cell requirement and efficiency planning. It enables the bottleneck cells to be identified for each different production route and also the number of components that FMS can produce to be determined for each production period.

The way in which the model functions is to consider all the production cells used, in any order which the user has defined for the production of a particular batch. Currently the Durham FMS can only process a single batch of any size at a time and once the order of cells to be utilised for a specific batch has been defined by the user it cannot be changed during the period of the batch production. However, the overall control software for the Durham FMS has the potential for further development to allow for batch mixing and job processing with variable production routes.

The mathematical model calculates the requirement, output rate and defective rate for all the cells which are to be used for a specific batch. The formulae employed to calculate each of these are explained in turn. The determination of the FMS cell requirement is carried out by successive use of the following formula:

$$Requirement = \frac{No.\ of\ loaded\ parts \times Time\ worked}{60 \times Efficiency \times Shift}$$

Where:

Requirement= the FMS cell requirement, i.e. number of specific cells is required.

Shift = the duration of an operating period defined by user in hours.

No. of loaded parts = the raw parts required per cell per shift.

Time worked = the processing time per part per cell in minutes.

and the FMS cell efficiency is calculated as follows:

$$Efficiency = 1 - \frac{Setup\ time + Downtime}{Analysed\ period}\%$$

Where:

Analysed period = the time utilised for the evaluation of the average setup and downtime values in hours (i.e. total period).

Setup time = the time spent for part mounting or pallet changing (if one exists) and is given in minutes per cell per analysed period of time.

Downtime = the time which the cell is not doing productive work (e.g. because of breakdown or maintenance) in minutes per cell per analysed period of time.

To estimate the number of raw parts required per production cell an assumption is made that defective parts cannot be salvaged and all parts in a batch must visit all the cells in the current production route. Although the defect rate for FMS are very low, it is a common practice to calculate the total number of parts produced on the FMS for each cell in the given production route as this considers that because of defective parts, there are less and less good parts transferred to the next cell. The following formula is used to obtain this.

$$Total\ part\ requirement = (Defective\ parts + Good\ parts)$$

The defect rate is also calculated for each successive cell in the selected production route and it is assumed that both the number of defective parts and the total number of parts are measured during the same analysed period of time. The formula used to calculate the defect rate is as follows:

$$Defect\ rate = \frac{Number\ of\ defective\ parts}{Total\ number\ of\ parts}$$

To satisfy the demand for the "good" parts during a shift the initial number of parts to be loaded to a cell the following formula is used.

$$No.\ of\ loaded\ parts = \frac{Good\ parts}{1 - Defect\ rate}$$

The algorithm calculates this value successively for each cell, following the selected production route. The functionality of the algorithm can be demonstrated through an example which shows what information could be provided to the user if the system was asked to produce a given number of parts. The example chosen here is for the production of 75 parts where each part in turn has to visit three production cells namely sawing, turning and milling. The module which performs this is within the MASTER program (MASTER-S1, Appendix F) . The information required of user entry is the number of finished good parts required, the length of the production period and the analysed period of time.

Figure (4-2) shows the output for FMS cell capacity check for a job namely "JOB5". The analysed period was chosen to be 8 hours and the system was asked to calculate for production of 75 good parts with a production period of 2 hours. The capacity program utilises the manufacturing database to obtain further information about the particular job. This information is then shown on the input data section of each of the cells, Fig. (4-2). The algorithm analyses the capacity needs and shows cell loading levels. The results show that the system requires more raw parts than the volume of the "good parts" and that is due to the non-zero defect rate of the cells. There exists one FMS capacity check output for each cell and in the calculated result section the following are computed; the number of raw parts required per production period, this number is different for each cell depending on the defect rate of that cell; the cell's efficiency; the adjusted equipment requirement which is equal to one for all the cells at the start and finally the unadjusted equipment requirement. The unadjusted requirement indicates the cell utilisation levels for the selected part production route. As the example indicates, since the turning cell has a higher setup and downtime, and also the processing time per part is longer it is less efficient than the other cells and more of this cell is required to be able to cope with the capacity requirements.

So far what has been described in this section only deals with the planning side of the part production and for real time control of the FMS a module namely Production Activity Control (PAC) is required.

4.4 Production Activity Control

Production Activity Control (PAC) is often seen as the execution of the long term plans developed from the master production schedule and materials plan at the production cell level. It plays an important role in linking the shop floor with the other elements of the Production Planning and Control (PPC) system [Lyons 1990]. Production plans received from the higher levels are transformed into control commands for the production process. Another role of PAC may be the translation of data from the shop floor into information which is useful to the higher level planning function in the PPC system.

The essential building blocks of Production Activity Control are: scheduler, dispatcher and monitor. The following description of these three highlights the important principle that decision making is passed down to the point where most knowledge and information is available.

4.4.1 The Scheduler

The scheduler develops schedules for each individual production cell based on the known manufacturing process routings and expected available capacity from the list of required orders produced by the MPS and MRP systems. The goal is to schedule activities so that only what is actually required is produced when it is needed and in the correct quantity. To achieve this the scheduler may try to optimise to several criteria, e.g.:

- the machine tool's idle time, machine tool's setup time, the average time that products are in the system, and the number of products waiting in the system for an operation must be minimised.
- the throughput of the system must be maximised.

4.4.2 The Dispatcher

The dispatcher can be viewed as a real-time scheduler which executes the final order of the job sequence for the workstations of each production cell. The dispatcher forms its orders on the basis of information received from the scheduler (i.e. the schedule) and the current status of the system. The information required includes: the workstation schedule; the routing data on each part; the workstation status; the part transporter status and the part locations.

An additional function of the dispatcher is to coordinate the individual workstation schedules and material movement control. However, the schedule is based on assumptions concerning the duration of operations, and not all events in the system will take place in exact accordance with the scheduler's assumptions. For example, due to unexpected problems, operations may take longer than the anticipated time. This could lead to unnecessary discrepancies of time between the availability of product and the readiness of the next workstation. It is up to the dispatcher to make intelligent decisions for as precise an execution of the schedule as possible. The dispatcher has to minimise the system disruption in the event of a local problem. Should a disturbance occur, it is the responsibility of the dispatcher to resequence jobs within the constraints of the given schedule. However, if the predicted schedule differs greatly from the real system status the dispatcher may request a new schedule from the scheduler.

4.4.3 The Monitor

The monitor collects data of the events which have already taken place such as process times, material availability, part status, scrap data, failure and downtime data etc. It then translates this data into information which is fed back to the scheduler and dispatcher and on this basis the decisions for the real-time control of activities are made. Such a data collection system has to be reliable, accurate and quick. The three building blocks of Production Activity Control can provide the basis for real-time scheduling and control of shop floor activities. In developing a scheduling system for the shop floor different techniques are possible and these are discussed in the next section.

4.5 Shop Floor Scheduling

One of the vital steps in planning and controlling the production environment is the scheduling of the production of orders. Scheduling may be defined as the assignment of jobs or work orders to production cells and workstations on the shop floor for a given period.

Typically, there exist several jobs awaiting processing at any given time and their routings depend upon the availability of required resources and the relative importance of time and cost. The dynamic variation of factory status can mean unpredictable changes with which the scheduling system must deal, usually resulting in rescheduling requirements.

The problems involved in the scheduling of a FMS are more of a general case of job shop scheduling problems, although factors such as additional resources constraint and real time operation makes FMS scheduling more complicated than classical job shop scheduling [Rabelo 1989].

In the operation of a scheduling process five key components, which must be closely coordinated, are identified on which a proposed methodology is based [Maley 1988]. These are: guidance from a historical knowledge base, forecasting of imminent occurrences, real-time feedback from the operation of the facility, direction of objectives and constraint from management and world data and improvement of the process from a planning module. The proposed system utilises real-time feedback from the operating facility, direct feedback from a simulation of the facility and guidance from a historical knowledge base to optimise the control of an automated manufacturing system.

Traditional Operational Research (OR) tools such as Linear Programming (LP) have been combined with queueing network models in an iterative procedure to make optimal use of the part mix and routing mix flexibility of the FMS [Avonts 1988]. The LP model determines a static solution based on operation times and available resources capacity. This solution then undergoes a queueing model examination in order to evaluate some dynamic effects. Finally, a more realistic solution is achieved by feeding this information back into the LP model.

Shop floor scheduling problems can be solved by distributing the decision making and scheduling tasks. The approach taken by the distributed scheduler is essentially a multi-agent problem-solving method [Shaw 1988]. Because of its complexity the scheduling problem is solved collectively among the production cells. It is considered to have two scheduling levels. The first level assigns jobs to the most appropriate cells and the second level scheduler executes intra-cell scheduling. Distributed scheduling utilises a bidding mechanism to coordinate the execution of tasks among the production cells. In this case a cellular manufacturing system in which cells are connected to a LAN. When a cell completes an operation (task), it announces a request for another cell to process the remaining operations. Each cell sends a message to indicate the Earliest Finishing Time (EFT) of the candidate task if processed in that cell. The cell controller assigns the task to the best bidder.

A scheduling algorithm has also been developed for Just-In-Time (JIT) production systems [Miltenburg 1989]. A JIT production system involves the simple concept of controlling the production of parts so that parts are only produced at one workstation such that will be available at the next as and when they are needed [Vail 1988]. One objective of a JIT system is to achieve a constant flow of parts with a minimum of waiting time between operations, and thus a minimum Work-In-Progress (WIP) inventory.

In addition Artificial Intelligence (AI) techniques are being used to solve shop floor scheduling problems [Bel 1989], [Copas 1990], [Sarin 1990], [Sepulveda 1988]. Several AI disciplines such as distributed decision making, data interpretation, optimisation and constraint-search may have to be utilised in order to provide a real-time solution to the scheduling problem [Rabelo 1989].

Although many of the aforementioned developed scheduling systems provide solutions to the scheduling problems, often these are based on several assumptions and may bypass some of the shop floor problems such as machine breakdown and maintenance, batch splitting, material availability etc. [Gupta 1989]. An interactive scheduler combines the human expertise with computing power as a decision support system to overcome this problem [Jackson 1989]. It utilises the principles of socio-technical design to achieve a "best match" solution. The human interaction provides the necessary input to generate a solution which more closely models a particular system and the computing power deals with the bulk of the tedious scheduling tasks.

For the purpose of this research a real-time scheduling system has been developed which is not as elaborate as some of those have been mentioned in above and this was due to the restriction that was imposed by the available system hardware to this research i.e. a network of BBC microcomputers (model B).

4.6 Real-time Scheduler for the Durham FMS

In the Durham FMS the process of part production begins once the user executes the "MASTER" program. This is in the form of a menu driven module, which provides the user with several options, including processing of a job. Choosing this option the user is asked to enter the name of a job together with the number of parts required (i.e. the batch size). This information is then passed to the scheduler module (SCHEDULER program, Appendix F). The scheduler resides at the master level, and is in effect the shop floor supervisor. It incorporates the database module together with its scheduling algorithm, and partially utilises the hierarchical behaviour of the system.

In developing the scheduler the following assumptions were made. The available production cells are dissimilar in their functionalities i.e. an assembly cell cannot perform a machining operation. Each production cell has its own cell controller. A job may have to visit several cells before reaching its final stage of processing. Production of any number of parts belonging to a particular job type is possible i.e. jobs can have any batch size. The route or possible routes of part production for any job is already defined in the manufacturing database. Jobs are processed one at a time. Defective parts cannot be salvaged. The FMS is equipped with a warehouse for raw material (parts), finished and defect part stores. Parts can be passed from one cell to any other cell according to the part production route for that job by means of an inter-cell part transport system (e.g. an AGV).

Once the system control is passed onto the scheduling module it initiates the operation of the system in two phases. In phase one, it searches the manufacturing database to obtain the relevant information regarding the job and the required manufacturing cells. It then downloads the Cell Controller Specific Software (CCSS) to the participant cells, and runs them remotely. The scheduler sends messages and control commands to the cell controllers, where they are translated. Each cell controller receives the name of a job and the number of parts to be manufactured from the scheduler. Communication between the various control levels of the system, (i.e. Master, cell, equipment), takes place via the `CellTalk` module which is explained in chapter five.

In phase two, with information about the sequence of operations and cell utilisation from the database, the scheduler executes the following tasks until the production of parts is completed:

a. supplying the cells with the parts from the previous cell as defined in the sequence,

b. if the cell is the first cell in the sequence, a raw part from the warehouse is sent to it,

c. if it is the last, parts are sent from that cell to the finished goods store.

The scheduler incorporates `CellTalk` to scan the progress of part production in all of the active cells for the current job, and if it finds the operation on a part is completed in a cell, it transfers that part to the next logical cell (which is not necessarily the next physical cell). The scheduler keeps a record of the number of parts taken from the warehouse. When this number is the same as the desired batch size to be manufactured, it continues to coordinate the part transfer between the cells until the batch is fully processed. If a defective part is detected in a cell by the scheduler, it sends that part to the defective part store, and fetches a raw part from the warehouse for the defective part replacement.

The scheduler does not attempt to prepare an operation schedule for the system resources, rather it allocates parts (and operations) to the cells during the actual production period (in real-time), utilising the System Monitoring Module (`SMM`) before making decisions according to a set of predefined rules and the availability of the cell for acceptance of the part. In this way, the scheduler does not need to make assumptions about e.g. the length of operation in each cell for each part. The operation of the scheduler is similar to a simple dispatcher which is capable of making decisions in real-time as and when the behaviour of the system changes, (i.e. a defective part may occur at any cell).

As parts are being processed in different cells the scheduler records the occurrences time of the system activities together with their types in real-time for off-line analysis. Information gathered here will provide input data for the System-wide activity chart discussed in chapter six.

After the successful processing of parts in each cell, the scheduler sends control commands to the cell controllers to end their part production session. The user is allowed to terminate the process of part production at any time, and this can be done at the master level. Once the scheduler receives instructions to terminate part production, it issues commands to the cell controllers, which are actively processing the current job, to terminate the operations of their cells. Each cell controller will then terminate the cell operation if and only if the current operation has been completed. In this way, the uploading of intermediate data is prevented during the termination phase of production.

The heterarchical behaviour of the system can be highlighted at the cell level, where the operation of each cell is autonomous and once the part production has started, each cell controller will take charge of the cell, and does not require any further commands from the scheduler to perform the in-cell operation. Every time a cell completes the operation on a part, it will indicate its availability to operate on further parts and will pull another part from the previous logical cell. If a failure occurs in a cell it need not have an immediate impact on the operation of the rest of the system. If another cell exists which can perform the required operation, then the faulty cell can be bypassed, and the system can continue in operation. This information interchange is done at the cell level through `CellTalk`.

The scheduling system has a potential to be utilised in a bidding mode, for the system with machining centres, or cells with similar functionalities. The scheduler will then request bids from different cells. Each cell will compute when it would be ready to accept a job, and a job is allocated to the cell with the shortest bidding time.

The scheduler module does not optimise the productivity of the FMS resources (machine tools) by changing the sequence of operations. Furthermore, no attempt is made to perform multijob processing or batch mixing. Appendix C provides a detailed description of the scheduling system.

The control and scheduling of activities within each production cell is performed by the Cell Controller Specific Software (CCSS) which is examined more closely in the next section.

4.7 Cell Controller Module and Scheduling of the Intra-cell Activities

In general, the expected tasks of a cell controller may be divided into the following groups; to control, to monitor and to report. A cell controller should be capable of to receiving work orders from levels two and three of the PPC system or a plant scheduling system. It must also be able to start, stop and pause machine tools send alarms, indicate status, give reasons for failure and give the user the ability to intervene with the system [Bertok 1989]. The definitions of cell control vary by industry, company and systems integrator and a distinction between cell monitoring and cell control has to be made. National Electrical Manufacturers Association (NEMA) in the USA is currently making an attempt to arrive at a set of standard definitions for cell controllers [Labs 1989]. At the moment, a large number of cell computers perform monitoring functions as opposed to control functions and do not operate in or near real-time [Larin 1989].

The functionality of the cell controller in the Durham FMS is defined as an entity which is in charge of monitoring, control and coordination of intra-cell operations. BBC microcomputers (model B), being the only hardware resources available for this research project, were to be utilised as an experimental set up to exhibit the practical aspect of designed modules. Since the programmable devices in each cell of the FMS are made by different manufacturers, they require different control signals for the same function throughout the cells (i.e. the control signal to 'STOP' the milling machine in milling cell is different from 'STOP' in sawing cell). Hence, there exists a need for a common interface to unify the meaning of control commands. A general purpose cell controller software is developed to produce Cell Controller Specific Software (CCSS). It contains common purpose routines such as INITIALISE, START, STOP, RESET, DOWNLOAD, UPLOAD, etc., and all the user has to do is include the control signals within the routines. This process is performed at the system definition stage, and a CCSS is required for each cell. New CCSS is created at the master level, but once in operation it resides at the cell level.

The CCSS has the role of a cell supervisor in the FMS and its functions can be divided into four groups;

a. issue control commands to the programmable devices,

b. coordinate the intra-cell activities,

c. generate production progress report,

d. communicate with other entities (i.e. cells and the system scheduler).

Once the process of part production begins, each cell controller receives the name of a particular job and the number of parts to be manufactured, from the system scheduler. It then utilises the manufacturing database to obtain the related job and cell information. From now on the CCSS of different cells communicate directly to one another to gain knowledge about the parts currently being entered to their cells. A CCSS may require information such as part orientation and part dimension etc., directly from the previous logical cell, which has just completed operation on the part. Each CCSS is capable of checking the status of part production in any other cell directly at any time. An important feature of the CCSS is that cells can interrogate each other directly without reference to the master level. This direct intercommunication between cells leads to the term "heterarchical" control.

Operation of CCSS can be described as follows: perform a monitoring cycle, if there is a status change check it against the predefined rules before performing the task and issuing the commands to the cell resources (i.e. programmable devices, machine tools etc.), Fig. (4-3).

During each monitoring cycle CCSS will check for the following; whether there is a message to be received from other entities, a part to be transferred to or from the cell, production of parts to be terminated and finally, whether or not the current operation under progress has been completed. This is done by CCSS scanning through its dynamic database (CAMP) to identify change of status in various memory locations within the CAMP (described in chapter five). Every time CCSS performs a task, for example, receives a part from other cells, it updates the status of the appropriate memory location (i.e. input part buffer in this case) within the CAMP. This feature will allow the cell dynamic information to be updated without the necessity of passing information over the network, and at the same time make this information available to any other entity. The monitoring cycle is executed every time a task is completed.

CCSS makes decisions only after a monitoring cycle has been executed, and its monitored condition checked against a set of predefined rules. The rules are defined in the form of a series of conditional statements. For example, if the monitoring module identifies that the current operation (e.g. machining or assembly) has been completed, criteria such as the state of output part buffer, being not full has to be met before the decision of part transfer from machine's table to output part buffer can be made.

Currently, the rules are defined and placed at many places throughout the CCSS program, Appendix F. It was possible to develop modules which would search for rules in a database but which needed decisions to made in real-time. A search in a database with computing facilities such as BBC micros would have slowed the process of decision-making. A detailed description of the CCSS program is included in Appendix D.

4.8 Conclusion

This chapter was concerned with the introduction of an architecture for Production Planning and Control. A module was developed which allowed the user to analyse the system capacity on a simulation basis and to identify the level of cells' utilisation for a job with a specific production route. Currently, this module can only consider the processing of a single job with a user defined production route, however its benefits can be enhanced once it has been developed further to cater for multi-job with variable production route. Furthermore another improvement may lie within the integration of this module to the system scheduler, obtaining on-line capacity information from it for maximising utilisation of the resources within the FMS.

A real-time scheduler was developed which controls the overall activity of the FMS. It reacts to the changes of the real system and issues commands to the cell level thereby exhibiting the hierarchical behaviour of the system. The scheduler utilises CellTalk for its communication with other entities and records the activities of the system in real time. The control and sequencing of the activities within a cell is carried out by the cell controller module. Once the production cells have started their operations, they will continue to operate autonomously and exercise the heterarchical control methodology by directly exchanging information with one another. This simplifies the scheduling task of the system as the scheduler does not need to schedule the in-cell activities. This has two advantages: firstly the decision making is located where the data is originated and secondly the volume of global information is reduced.

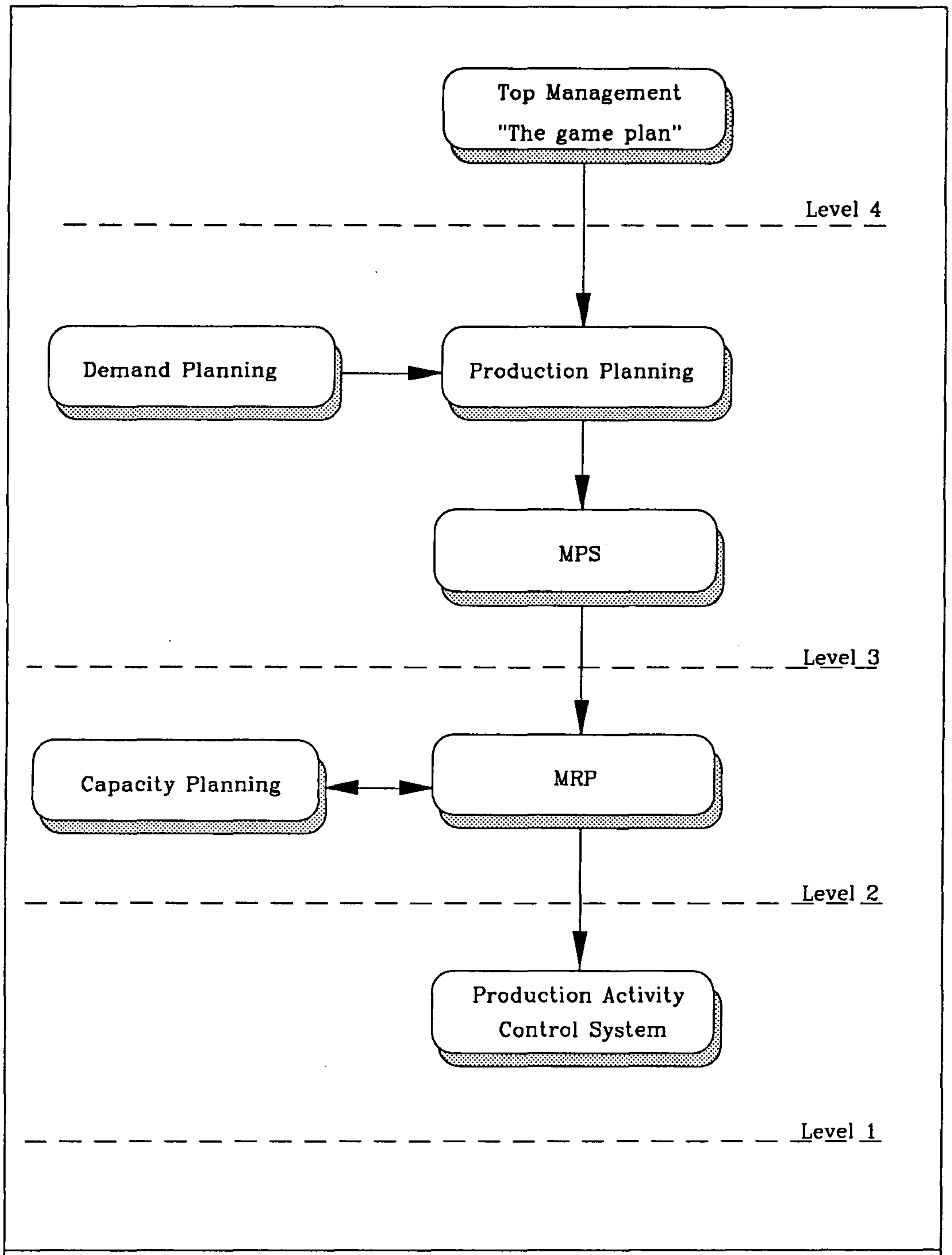


Figure (4-1) Production Planning and Control Structure.

```
          ***** SAWING    cell *****
Input data
----------
Finished part required per production period  =75.00
Length of production period                   =2.00 Hours
Analyzed period of time                       =8.00 Hours
Average setup time                            =0.20 Mins
Average downtime                              =1.00 Mins
Processing time per component                 =0.50 Mins
Defect rate of the cell                       =4.00 %

Calculated results
------------------
Raw parts required per production period      =81.00
FMS cell efficiency                           =99.75 %
Unadjusted equipment requirement              =0.33 FMS cells
Adjusted equipment requirement                =1.00 FMS cells
          ***** TURNING   cell *****
Input data
----------
Finished part required per production period  =75.00
Length of production period                   =2.00 Hours
Analyzed period of time                       =8.00 Hours
Average setup time                            =2.50 Mins
Average downtime                              =3.00 Mins
Processing time per component                 =3.12 Mins
Defect rate of the cell                       =1.18 %

Calculated results
------------------
Raw parts required per production period      =78.00
FMS cell efficiency                           =98.85 %
Unadjusted equipment requirement              =2.00 FMS cells
Adjusted equipment requirement                =1.00 FMS cells
          ***** MILLING   cell *****
Input data
----------
Finished part required per production period  =75.00
Length of production period                   =2.00 Hours
Analyzed period of time                       =8.00 Hours
Average setup time                            =0.80 Mins
Average downtime                              =1.10 Mins
Processing time per component                 =1.00 Mins
Defect rate of the cell                       =3.00 %

Calculated results
------------------
Raw parts required per production period      =77.00
FMS cell efficiency                           =99.60 %
Unadjusted equipment requirement              =0.65 FMS cells
Adjusted equipment requirement                =1.00 FMS cells
```

Figure (4-2) FMS cell capacity check for JOB5.

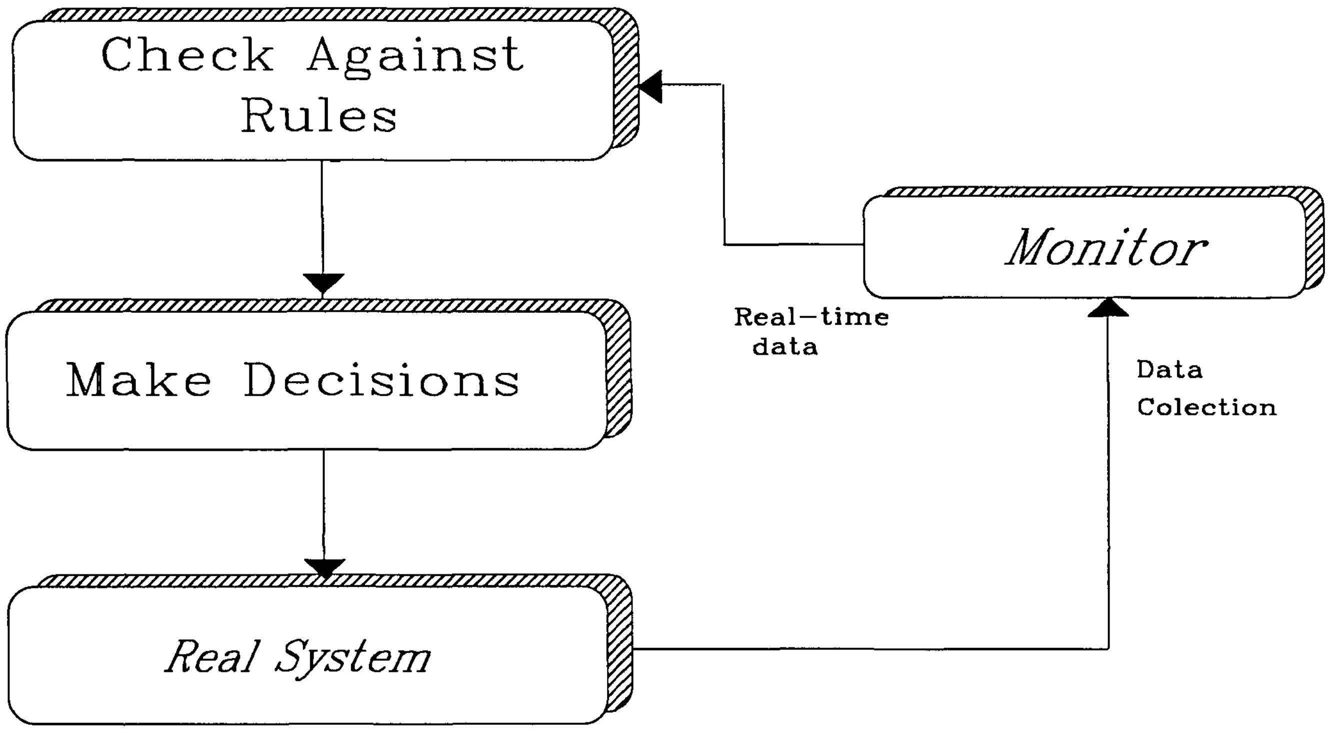


Figure (4–3) The cell controller operations.

CHAPTER Five

Communications in Flexible Manufacturing Systems

5.1 Introduction

Today's Flexible Manufacturing Systems are composed of various programmable devices such as numerical controllers (CNCs), robots, programmable logic controllers (PLCs), etc. Effective use of these resources requires that they be interconnected by a communication system. Utilising Local Area Networks may reduce the cabling problems and improve access to distributed information while at the same time increasing the flexibility of the system.

Currently, one of the problems with which many manufacturing companies and system designers are faced, is the introduction and integration of equipment from vendors other than the one from whom they have purchased their existing equipment. Integrating multivendor equipment requires custom built software and hardware which in turn leads to a high expenditure. As a result many smaller manufacturing companies, once they have been set up, may suffer from being unable to choose the best equipment from a variety of vendors and being stuck with the product of a single vendor. This chapter highlights the problems related to the intercommunication of systems within a manufacturing environment and how these problems may be solved. Section 5.2 points out the importance of the open system and how the manufacturing environment can benefit from it. In section 5.3 an introduction, the role and different types of Local Area Networks are introduced. Section 5.4 talks about MAP, a solution proposed and developed by General Motors. The collapsed communication architecture, Enhanced Performance Architecture (EPA) and miniMAP are discussed in section

5.5, this is then followed by a description of the LAN utilised for Durham FMS in section 5.6. A communication technique, namely CellTalk, allowing the direct communication between entities in the FMS was developed and implemented on the experimental Durham FMS and is discussed in section 5.7. CellTalk takes advantage of a hierarchical structure for information exchange among entities in the FMS and incorporates a series of communication primitives which are described in section 5.8.

5.2 Manufacturing Communications and OSI

There have been fundamental changes in the application of automation to the manufacturing industry in the past two decades. This is due to the application of computers to the control of the equipment and processes involved. As the number of programmable devices within the manufacturing organization grow, the lack of communication standards leads to the frustration of those wishing to integrate the operation of such devices in the interest of manufacturing efficiency [Sumpter 1985], [Street 1989].

System designers who build computer based automated systems often must decide how to integrate effectively system modules into one system. One way of achieving this objective has been to choose all the equipment from the same vendor. Although all the equipment from one vendor can communicate with one another, such communication is considered to be "closed" since there exists a need for a special custom designed protocol convertor to allow communication with equipment of another vendor if implemented. Alternatively, an "open" communication standard can overcome the interconnection problem, because all the vendors will develop their systems according to the same standard, which defines all types of information transfer between various elements of the systems [Weston 1986].

As it would be extremely difficult to arrive at a general description of the communication process which covers the behaviour of every type of communication system, an international standard model, namely Open System Interconnection (OSI), is necessary [Zimmerman 1980]. The prime task of OSI is to overcome the communication problem between computers and associated devices, so that each is able to exchange information with the other regardless of make, model, age or level of complexity [Tangney 1988].

One of the factors responsible for bringing OSI to its present stage of development was the realisation of major manufacturers of computers and programmable devices (intelligent entities) of the internetworking problems, resulting from the development of different communication architecture by each manufacturer [Freer 1988]. The disadvantages that these manufacturers were foreseeing included; the user discouragement due to problems involved with compatibility of their systems and the threat to the size of the communication market, the attraction of users to the biggest manufacturers. From the user's point of view, it was a great danger to be limited to one supplier for their communication requirements. As a result the International Standard Organisation (ISO) proposed a standard reference model to provide a framework so that standard protocols could be developed.

The OSI reference model breaks down the network and support services into a modular fashion producing a seven layer hierarchy shown in Figure (5-1). The model is concerned only with the external behaviour of computer systems and the communication's functions are partitioned by it into the seven layers [Zimmerman 1980]. An important part of this partitioning is that it separates communication-oriented functions at the lower layers from more user/application-orientated functions at the upper layers. The communication-oriented functions are concerned with the use of the physical data transmission network and the rules associated with its use are the low-level protocols. The user-oriented functions are concerned with the ability of two systems to exchange information and to understand it. The rules required to achieve this cooperation are called high-level protocols.

There are two types of standards associated with each layer of the OSI reference model namely, service specification and protocol definition. The service specification [Beauchamp 1987] of each layer (layer N) defines the facilities and functions offered to the layer immediately above it (layer N+1), and the layer N, in turn, uses the services provided by the layer below (layer N-1), Fig. (5-2). The protocol definition of a layer (layer N) defines the rules and conventions for communicating with a corresponding layer (layer N) on another system.

Note that two systems, each of which can rightly claim to conform to the OSI seven layer reference model, will not necessarily be able to communicate with each other directly. To accomplish direct compatibility there must be an agreed implementation specified for each of the seven layers [Weston 1986]. This is what the Manufacturing Automation Protocol (MAP) is aiming for and is discussed in section 5.4.

Since MAP is about networking and is mostly concerned with Local Area Networks (LANs), which are the private channels used within the manufacturing environment, [Wright 1987], some concept and terminology of LANs is discussed first.

5.3 Local Area Networks

A Local Area Network (LAN) is primarily a data transmission system, intended to link computers and associated programmable devices within a restricted geographical area [Frenzel 1987]. LANs are used in industry at several different levels which may serve different purposes. Thus it is not surprising that the most efficient type of networks for one application may not be the most suitable for another. What is important is the intercommunication ability of different network with the requirement for minimum effort.

Office automation and factory automation are the two major areas in which LANs are highly valued [Hohner 1989]. Some of the particular benefits that LANs are capable of offering to office automation are: resource sharing such as mass storage, gateway, high quality printers; users can have easier access to information and information stored at any one point can be accessed by all those who have the need and the authority; most LANs can be reconfigured, changing both cable length, the position and number of devices attached to them. Factory automation is concerned with the interconnection of devices for the control and monitoring of a manufacturing operation. Local area networks here will allow a distributed set of programmable devices such as sensor, CNC machines, robots, etc. to be monitored and the appropriate control action to take place if necessary [Allan 1989].

A wide variety of relevant factors may be used to differentiate the many types of LAN available today i.e. reliability, price, performance, the maximum size of the area they can cover and so on. To best describe a particular LAN design [Hutchison 1988] from a technical point of view, one must adopt the approach taken in each of the following technological areas; network topology, transmission medium, signalling technique and accessing method, which are examined in turn.

5.3.1 Network Topology

The topology of a network describes the way in which the stations (nodes) of the network are interconnected and gives the logical shape of the network. The star, ring and bus are the three basic LAN topologies.

In the star topology all the stations (nodes) comprising the network are connected to a common central switch, Fig. (5-3a). The central switch provides a path between any two stations, either logically in a packet switch, or physically in a circuit switch. This topology exhibits a centralised communication control strategy. Failure of the central switch will stop all communication in the entire network. Throughput is limited by the rate of acceptance of the central switch, which may form a bottleneck.

A ring network consists of a number of stations linked to its neighbour by a point-to-point unidirectional link via a repeater to form a closed ring [Ross 1987], Fig. (5-3b). The data is sent in packets or frames around the ring in one direction from station to station. Each packet contains a destination address. Within each station there exists a controller board responsible for access control to the ring and the packets addressed to that station.

Repeaters carry out three main functions which are; data insertion, data reception and data removal. As the packet circulates past a repeater, the destination address is checked, if it corresponds to that station to which the repeater is attached, the remainder of the packet is also copied before being regenerated and retransmitted bit by bit within the packet frame to the next/another repeater on the ring.

Throughput is determined by the media and the capability of the repeaters. Techniques available to access a ring network include the slotted ring, which involves circulating a number of empty data packets around the ring, and the token ring in which short, unique data packets may be exchanged for much longer data packets.

Bus networks have a very different approach than ring and star. These are comprised of a shared linear communication medium (bus) to which stations are attached by special hardware interfacing or tap to the bus, Fig. (5-3c). Each tap must be capable of delivering the signal to all stations in the network. Data from one station is transmitted in packets to all other stations on the bus in both directions at the same time. Each station listens to the transmission and picks up the one to which it is addressed. If the active component of one station fails, it does not effect the rest of the network. For this reason bus networks are often referred to as "passive".

5.3.2 Transmission Medium

The transmission medium is the physical medium linking the stations (nodes) in a LAN. There are two main types of transmission media; terrestrial media and free space. Transmission through free space may be by digital, microwave, radio and infra red light beam communication. Terrestrial media include twisted pair wire, coaxial cable and fibre optic cable, which is widely used in commercial LANs. A twisted pair wire consists of two insulated wires arranged in a regular spiral pattern. Twisted pair wire is suitable for both analogue and digital data transmission [Freer 1988].

Coaxial cable consists of a conducting wire inside a second, hollow conductor and separated from the hollow conductor by an insulator. The hollow conductor is earthed and shields the signals on the central conductor from interference. Two modes of data transmission are possible using a coaxial cable: baseband and broadband, which are explained in the 5.3.3 sub section.

Finally, the fibre optic cable is a thin flexible strand of glass or plastic (typically 0.2mm in diameter) surrounded by a PVC jacket capable of conducting an optical ray. Its advantage over traditional electrical transmission media includes; low attenuation, very high bandwidth, immunity to electrical interference and crosstalk.

5.3.3 Signalling Technique

There are two major classes of signalling technique employed by LANs, namely: baseband and broadband. A baseband system uses digital signals on single frequency, (i.e. the signal is transmitted on its original frequency without any modulation) in half duplex mode only (i.e. in one direction at a time only). Transmission media such as twisted pair, coaxial cable and fibre optic can be used for baseband signalling.

Time Division Multiplexing (TDM) techniques may be used to allow multiple messages to travel simultaneously on a baseband medium. Stations are attached to the transmission medium by means of taps. A terminator may be used to prevent signal reflection on twisted pair and coaxial cable.

In broadband systems analogue signalling is used, in Radio Frequency (RF) range, and coaxial cable is employed as a transmission medium which can support transmission over a wide range of frequency (generally bandwidth of up to 300-400 MHz). These analogue signals are also known as carrier signals. The bandwidth of a transmission medium can be divided into narrower bands (channels) by Frequency Division Multiplexing (FDM) and each can be utilised independently to provide a range of services. The available channels on a broadband system are logically independent of each other and may utilise different accessing methods and operate at different speeds.

Each station requires a modem to modulate and demodulate the digital signal onto the carrier signal for transmission over the network. Frequency agile modems are special types of modems which can operate on several different frequencies, and are capable of being switched locally or remotely, from one to another. Unlike the baseband, however, broadband is a unidirectional medium. Signals inserted onto the medium can propagate in only one direction so separate transmit and receive channels must be provided. There are two methods for configuring broadband networks: Single cable and dual cable approaches. In a single cable system, often referred to as a mid-split system, the bandwidth of the media is split into transmit and receive band with a guard band in between. The guard band prevents interference between the transmit and receive frequency bands. Signals amplifiers separate and amplify the two bands in different directions and a headend is used to receive the signals in the transmit band and retransmit them on the receive frequencies, Fig. (5-4a).

In a dual cable system, one cable is used to transmit information while the other is employed to receive the information. Here the entire bandwidth of the cable may be used for data transmission, and the two cables are simply looped around at the headend. Each station is connected to two cables transmitting all data on one cable and receiving on the other, Fig. (5-4b). In the broadband system Time Division Multiplexing (TDM) may be used to allow several users access to a channel in quick succession.

The advantages offered by broadband system are many, including: the ability to carry a wide variety of traffic on a number of channels (voice, video and data channels); with the use of amplifiers a wide area of coverage is achieved; channel bandwidth allocation may be changed to match changing requirements.

The disadvantages include: the average propagation delay between stations for broadband is twice that of a comparable baseband system. This may reduce the performance and efficiency of the system. There is a risk of total failure, due to a short circuit on the cable or failure of a headend . The cost of implementation of a broadband system is higher than other LANs.

5.3.4 Accessing Method

The way in which a station can get access to a LAN is defined by a set of procedures called access method or protocol. Some of the most popular accessing methods include the distributed access methods which employ some form of TDM. Here, each station obeys certain fixed rules in order to know how and when to gain access to the LAN. Token passing, empty slot and Carrier Sense Multiple Access with Collision Detect (CSMA/CD) are the most popular examples of distributed accessing methods.

The token passing can be implemented on both ring or bus LANs. In the token bus topology the stations form a logical ring and each station is aware of its logical successor and predecessor addresses. The operations of token bus and token ring are similar to each other. The right to use the transmission media is determined by a free token which continuously circulates from station to station in the logical ring (token bus) or physical ring (token ring) [Pitt 1987].

To transmit data a station awaits the arrival of a free token. It then marks the token as busy and sends a data packet. As the token has passed around the ring it arrives back at the sender where it is marked as free before being sent to the next/another station on the ring. Note that each station must stop transmitting and pass on the token after a set maximum time. This method gives a definable maximum time for data transfer and for this reason token passing LANs are said to be deterministic. The deterministic behaviour of token passing LANs makes them particularly important in the industrial real-time control and in analysing critical factory data flow.

The empty slot or Cambridge ring access method utilises a ring that is divided into a number of fixed size slots. A station wishing to transmit data waits to use a circulating empty packet (slot) as it passes by. It then marks the packet full and inserts its own address and data, and the destination address before passing it around the ring. The destination station recognises its address, reads the data, marks the packet as read and then passes the packet on around the ring. When the packet arrives at the transmitting station, it marks the packet as empty and the transmitting station is not permitted to reuse the packet twice or more in succession, hence the empty packet would be sent to the next station on the ring.

The CSMA/CD method of accessing relies on carrier sense to ensure that only one station is transmitting on the medium at any one time. A station wishing to transmit listens to the transmission channel to detect if any other station is transmitting. Since it takes a short time for a signal to travel along the transmission medium, it is possible for one or more station to begin transmission simultaneously, in which case a collision will occur. To prevent this and to detect any collision the transmitting station will listen during transmission as well as before transmission. Should a collision occur, the transmitting station will abort transmitting data, wait for a random period of time and will commence retransmission only if no other station has begun transmission in the meantime. CSMA/CD is restricted to bus LANs.

Local Area Networks are appearing more in the manufacturing environment where most of the computers reside within the programmable devices (machine tools) on the shop floor [Breeze 1990]. However, factory automation is still some way behind the computer industry in the process of standards' adoption. As a result the equipment on the shop floor lacks the power of "plug-in compatibility". General Motors (GM) has been trying to solve this problem by introducing its own set of standards called Manufacturing Automation Protocol (MAP) which is discussed in the following section.

5.4 Manufacturing Automation Protocol (MAP)

In any one factory automation project, somewhere between a third and a half of the total expense may be incurred in the special wiring and custom hardware and software interfaces required to enable the various programmable devices to communicate with one another [Dwyer 1987].

General Motors (GM) of the USA was one of the first companies to anticipate the scale of the ever-growing problem with communication among shop floor programmable devices in the manufacturing industry. By 1986, of 40,000 programmable devices, instruments and systems installed at General Motors (GM) facilities, only 15 percent were able to talk to one another. Hence, unless the idea of using a set of standard communication protocols was embraced by the manufacturers of various programmable devices, the problem of communication in the manufacturing industries would soon get out of hand [Gillespie 1988]. As a result, in 1980 GM initiated work on Manufacturing Automation Protocol (MAP).

MAP is a set of communication protocols, intended to be the standard basis for a factory-wide communication. It is an implementation of the OSI reference model which uses a set of selected standards from the OSI model and operates in LANs. The solution proposed by MAP, imposes a single, vendor-independent communication network standard on all manufacturers. Consequently, this will result in freedom of choice in selecting devices for the factory without special consideration for communication needs. Since the first release of MAP specification in 1982, it has evolved through several versions, the last version of which, 3.0, was released in June 1987 [MAP 1987], [Folts 1988]. Figure (5-5) illustrates the MAP 3.0 selection of standards within OSI model. By 1984 MAP dominated the interest of many manufacturers and users in the USA. One year later, the European MAP user group (EMUG) was formed in Europe, which revealed the interest of European manufacturers. The Communication Network for Manufacturing Applications (CNMA) project, as a sub project of (ESPRIT) i.e. the European Strategic Program for Research and development in Information Technology, is mainly concerned with CIM system applications and started at the beginning of 1986. One task of the CNMA project has been the definition of protocol profiles for communication services supporting CIM and to demonstrate internetworking of multi-vendor equipment benefiting those companies which are looking for low-cost multi-vendor MAP communication capabilities [Daigle 1988].

In order to eliminate any ambiguity towards the understanding of MAP standards, which may result in internetworking problems, everything from hardware to the meaning of exchanged messages is being specified within the MAP specification [Jones 1988]. As the layer seven (application layer) of the OSI reference model plays an important role in a MAP based network, it is examined more closely in Appendix E. Since the MAP products must be fully OSI compatible, MAP will have associated with each of its protocol layers a set of tests which will ensure absolute compliance with that particular OSI standard.

MAP discourages the use of bus protocols operating under CSMA/CD (e.g. Ethernet), because the non-deterministic nature of such a system may degrade the communication service to below the standard required by manufacturing systems. Hence, a broadband token bus system such as a logical ring with a data rate of 10Mbit/s has been chosen for a number of reasons including; the deterministic nature of logical ring, i.e. calculable maximum waiting time for a given station; the ability for communication of several devices simultaneously; reliable operation with high traffic loading, etc.

However, recent reports show that CSMA/CD Ethernet has become the dominant LAN for the manufacturing environment [Adams 1990]. It is stated that the success of CSMA/CD lies within the satisfaction of four basic requirements which are: cable media be capable of spanning shop floor and factory wide distances; good price performance on several cable media types; quick/easy access to the LAN channel under fluctuating network traffic loads; uninterrupted service in shop floor area where electromagnetic fields radiate from production equipment.

Furthermore, the head of the CNMA project at Aerospatiale [Communique 1990] also addresses the wide use of Ethernet as a common problem among many Europe manufacturers. He states that Ethernet is totally incompatible with US MAP specification and it is not possible to scrap all existing systems and start from scratch with a MAP compatible system. Therefore CNMA is attempting to find ways to incorporate Ethernet into an open systems communication hierarchy. Currently a gateway is used to allow factory control network (Ethernet) to communicate with the implemented shop floor carrier band (discussed next) MAP network.

Since the cost of the broadband system implementation is high, MAP specification intends to be media independent, giving the opportunity for a single channel MAP network to be utilised. This alternative MAP system is called carrier band MAP [Ioannou 1987] which does not require a headend remodulator. In carrier band MAP two frequencies are used, one to represent logic zero and the other, logic one, and utilising "Frequency Shift Keying" (FSK) treats the two frequencies as a single frequency and by shifting it up and down creates the zero and one signals. The single channel carrier band is, lower-cost and slower than the full MAP version, operated at 5Mbit/s. The carrier band MAP may be used at cell level of the manufacturing organisation to connect the various programmable devices within a cell, where multi-channels is not required [Coleman 1988].

Although implementation of full MAP communication spine, (often referred to as the backbone network), may seem ideal for covering great distances around the factory, it does exhibit some weakness for time critical applications. A typical assembly cell, which may be comprised a few robots and a vision system, requires simplified and rapid communication. The data transfer within the cell is often restricted to small amounts in order to obtain short messages at frequent intervals for real time applications. Hence, a full MAP system on broadband would both be expensive and very slow in terms of response time. The solution for such application lies within the use of a collapsed communication architecture namely Enhanced Performance Architecture (EPA) and miniMAP, on a baseband cable to provide fast responses [Weston 1987].

5.5 Enhanced Performance Architecture (EPA) and miniMAP

To arrive at a collapsed communication architecture is to dispose of some of the layers, Fig. (5-6a), which add data to the message as they pass through the OSI model. One way of achieving this would be to provide some means of direct link to layer 2, which would allow the application to communicate without going through the top five layers of the seven layer OSI model. A system which has on one side all seven layers and layer 1, 2 and 7 on the other side, is called Enhanced Performance Architecture (EPA). This allows communication on one side with its peer entity on the MAP backbone, as it contains all the seven layers and rapid data transfer on the collapsed side.

Some systems which only provide the collapsed architecture such as EPA with no seven layer architecture on the other side is called miniMAP, Fig. (5-6b). It must be stressed that miniMAP cannot communicate with an OSI system and is not OSI compatible. However, a miniMAP system may choose to communicate with an OSI system via EPA.

Finally, there is one more type of communication for which MAP may have to allow, namely fieldbus [Pleinevaux 1988]. Connecting every switch and sensor on the shop floor to a carrier band may result in high expense. Therefore, at sensor level there exists a need for a simple, low-cost and effective single link (fieldbus) which replaces the point to point links from each switch and sensor to its controlling equipment. The architecture of a fieldbus is comprised of layers 1, 2 and 7. The communication between the fieldbus and MAP on the carrierband requires a gateway. Figure (5-6c) shows a typical MAP network. It demonstrates the capability of a typical MAP network, satisfying all the communication requirements of various programmable devices within a plant.

However, for a small manufacturing company, neither would it be possible to invest money on the full MAP broadband, nor is there a need for multi channels. Hence, it seems logical to start with implementing EPA on carrierband MAP since it is both lower in cost than broadband and has the potential to become an integral part of a full MAP system later. Should there be a need for a multi channel broadband system in future, the existing communication network will be able to communicate with it via a bridge or router. To this effect there exists an opportunity for every manufacturing company, irrespective of their size and capital, to take advantage of the standard environment that MAP networks create, thus pressurising the manufacturers and vendors of computer-based equipment to move towards the 'plug-in compatibility' of their products.

It is believed that communication techniques which are network independent will have great value in the manufacturing industry. Therefore the research carried out for the Durham FMS is aiming at the development of a communication technique which may be applied to any type of baseband LAN as the starting point for the integration of resources within a manufacturing facility but at the same time allowing the system to have the potential to be integrated with the more industry accepted standard (i.e. MAP). Although MAP at the present time may not be implemented to many industrial systems as was first anticipated, in near future this will change once the MAP product becomes more widely available and also at lower cost. Hence, it is important that current efforts be put towards the preparation and, or adaptation of current systems to a well accepted standard and not the introduction of a further new set of standards. The way this can be achieved is by the development of a communication technique which solves the immediate problems in the manufacturing environment at a low cost (i.e. utilising the available network and resources instead of investing heavy sums in a full MAP system). Such an attempt has been made for the FMS rig at Durham. The network utilised as an experimental system to which the developed communication technique may be applied was a low-cost baseband LAN which existed prior to the start of the research. This LAN is called "Econet" which allows BBC microcomputers to be utilised as stations (nodes) in the network.

5.6 Econet a LAN for Durham FMS

Econet is a low-cost local area network for BBC microcomputers. This commercially available product allows all the computers on the network to share facilities such as printer and disc drive as well as the transfer of data between the computers. Only a minor addition to each BBC will enable it to operate on the network and that BBC is then called a "station". Econet uses an eight bit address which allows up to 255 stations to be interconnected over distances of up to 1 km.

Each station on the network only responds to a unique number (station id) which is hardwired onto its Econet interface card. The main cost of such a system is the wiring between the stations and the file server, a dedicated BBC to operate the disc drive. A station acting as file server cannot function as a user station at the same time. The file server software manages the files of all the users on the network. There are various levels of file server programs and level two which requires a 6502 second processor is used for Durham FMS. It supports a hierarchical directory structure. The route directory contains each user's directory which, in turn, can contain sub-directories. Simple protection facilities are provided to ensure that the user's file can be protected from other users.

Econet is a bus network using cable transmission over two pairs of wires which connect all of the stations together. Each station on the network is connected to the bus via a short spur. The network is terminated at each end by means of a terminator box which is a combination of power supply and circuits to prevent reflection of signals. Information travels in a serial bit form over one pair of wires and the other pair carries the network synchronization signal or "clock". The maximum rate of this clock is 210 KHz and if the clock is set for a transmission speed of 210 kbps (210K baud), a 64K of data block would monopolise the network for just under two and a half seconds. However, if a station needs to read data from a disc the effective data transmission rate is much lower than this. It takes 16 ms to read 256 bytes from a disc, and a further 6 ms for network transfer to occur [Cheong 1983].

Econet incorporates an access protocol similar to Carrier Sense Multiple Access with Collision Detect (CSMA/CD). The system will continue to operate if other stations are not working. The bus approach allows stations to communicate without involving a third party and the interface hardware is simple.

Econet takes advantage of a balanced line system [Napier 1984]. This means that the two wires of each pair always carry opposite voltages. When the signal goes positive on one, it goes equally negative on the other. The signal is detected by a differential receiver (i.e. one that operated only on the potential difference between the pair of wires). In this way two benefits are obtained as following:

a. in each twisted pair, the RF radiation from one wire cancels out the other. Radiated signals which could interfere with other electrical equipment are negligible.

b. external interference (e.g. inductive pick-up) is likely to affect both wires equally so will be ignored by the system.

The control of communication in the network is carried out by a system of interrupts. An interrupt is either generated by a local station's operating system or by the interface card when it detects the appropriate station number on the network. It then stops the computer's original task so that it can deal with the transmission or reception of data. These occurrences constitute a background task, the only effect of which, on the performance of a station, is a small drop in speed. Since the operation of one interrupt disables any other, the use of interrupts in the control of peripherals should be avoided.

Econet cannot provide the advantages offered by a deterministic type network since it utilise a contention method of accessing (CSMA/CD). This could have been a major drawback in its use for the FMS if all the elements of each cell (i.e. programmable devices) were attached directly to the network, Fig. (5-7a). Then the control of time critical operations by cell controllers would have been impossible. For example for a cell controller to stop the machining operation on a CNC due to a tool breakage, it had to wait until the network allowed it to transfer the appropriate control command to the CNC controller. As this could have fatal consequences a LAN with a deterministic nature (i.e. token rig or token bus) has to be employed to connect all the elements of cells to the FMS if above arrangement is used.

However if only the cell controller were interfaced to the LAN and all the devices within each cell were attached by means of a point-to-point connection to the cell controllers then the above problem is eliminated, Fig. (5-7b). The Durham FMS is arranged in this way where all the programmable devices are connected to their cell controller which they in turn are interfaced to the LAN. In this way each cell controller does not require the network to pass information to the in-cell devices. Therefore cell controllers could control the programmable devices within their cells in real-time and in a deterministic manner and not dependent on the volume of traffic in the network. Note that this method of FMS arrangement is dependent on the input/output capability of each cell controller resulting in a limitation on the number of programmable devices which could be attached to the cell controllers. This is where a token bus or ring MAP network would have an advantage over this method at the cell level. However in most cases depending on the definition of manufacturing cell they do not have more than a handful of programmable devices.

Econet was found to be entirely satisfactory for the laboratory based system. The communication technique developed for Durham FMS described in the following section is not LAN specific, therefore other architecture could also be employed.

5.7 Communication Technique

A communication software module, namely "CellTalk", was written to provide information exchange between different entities throughout the system. The aim of the technique developed was mainly to provide a solution for those manufacturing companies wishing to integrate multivendor equipment into their existing programmable devices, while at the same time leaving the option open for easy integration to a MAP based network. This technique could be particularly useful for those companies which possess earlier models of programmable devices (i.e. NC and CNC machines) that cannot take advantage of direct interface to a

LAN or a MAP system.

In general there exist two main solutions to the problem involved in the interconnection of shop floor devices [Weston 1989]. Firstly, a "backplane" solution which requires the incorporation of a network interface card into the hardware of the machine controller and consequently extensive modifications to the software structure of the controller. This solution is entirely dependent upon the cooperation of the programmable devices' manufacturers, however it will take a considerable time for such manufacturers to recognise the demand and provide such features as directly MAP compatible devices. Secondly, a "gateway solution" which can link those programmable devices that only support vendor specific protocols to the MAP based network. Hence, utilisation of gateways may provide an immediate solution to the problem of interconnection of shop floor devices.

In parallel to this research, the System Integration (SI) research group at Loughborough University [Weston 1989] has used gateways to connect together the elements of a manufacturing cell. These gateways were located in the vicinity of the programmable devices and acting as a front-end processor which provide the protocol conversion between the proprietary protocols and standard MAP network protocol. An example of such gateways were shown by SI in action at the CIMAP exhibition at the NEC, Birmingham, UK, in December 1986.

Although a gateway can provide an immediate solution, since it converts proprietary protocol to the standard MAP network protocol, it may slow down the process of direct information exchange among the entities. Therefore it was thought that a communication technique which could provide a mechanism for direct information exchange between the entities may benefit those entities requiring rapid data exchange. The designed and developed communication technique is called "CellTalk".

CellTalk uses a reserved area of memory for communication in each entity. This reserved area of memory has the same address in all entities and is therefore named CAMP (Common Address Memory in distributer Processors). Figure (5-8) shows the logical position of the CAMPs in the FMS. At the cell level each cell controller has a communication module comprised of CellTalk and CAMP which allows the information to be freely exchanged among entities.

Each memory location within the CAMP is given a unique name and a range of system states are held in coded form in this memory area. Assigning names to memory locations is carried out at the beginning of all of the programs which utilise the communication module. Each cell controller has one CAMP to which it writes the status of the cell activities. For instance, the content of memory location "program-stat", within the CAMP in the milling cell represents the equivalent information about the milling cell as of that in the vision cell (i.e. whether it is running, completed or stopped). In this way each cell controller avoids requiring to utilise the network to update the various cell states in its CAMP, as this is done at the cell local processor and therefore the volume of network traffic is considerably reduced. However, during the period of a data transaction between two entities on the network, entities must use the network as a means of data transfer.

Since the computers used as cell controllers are BBC microcomputers, they are set to operate in MODE 7 so that the area of memory beyond 7B00 (hex) can be reserved for CAMP. There are at least two hundred memory locations which could each hold a particular system state in the form of an encoded number (255 different binary states). Table (5-1) shows part of the listed memory locations used for holding cell states data. The table gives the information on the names and addresses of each memory location within the CAMP. It also shows what states can be written to or read from these memory locations and by whom. For example, an entity at master level, to establish a logical association with a cell controller, has to write a FF (hex) (for connect) into the memory location called "logi-assoc" with the 7B00 (hex) address. The cell controller will then read this location to find out whether or not an entity wishes to talk to it. Once the information exchange is completed between the entities, the logi-assoc will be set to 00 (hex) (for disconnect) by the entity at master level to signify the end of the session.

An alternative to the CAMP based system would have been to reserve an area of memory in a "common-memory" residing in the file server or in a separate entity on the network for each cell controller, Fig. (5-9a). In a common-memory based system, for a cell controller to update different cell states, it has initially to find the memory base address allocated to it in the common-memory, before passing the information over the network and writing it above that address. For example, cell controller 2, to update the status of the memory allocated for the programmable device state (i.e. whether it is completed, idle, stopped etc.), with indexed address of 0F (7C0F absolute address), it first has to find the base address which has been allocated to it by the system. The base address in this case is chosen to be 7C00 (hex). Once this information is obtained it uses the network to pass the information over to the common-memory before registering the desired state to the memory location. Note that the least two significant figures in the address represent the memory location addresses for cell specific data (status) and the two most significant figures of the address represent the specific cell base address in the common-memory.

In the CAMP based system, unlike the common-memory based system, the base addresses are the same in all of the cell controllers and the common memory is distributed in the cell controllers, Fig. (5-9b). For the cell controller 2 to update the device state in 7B0F address all it needs to do is write the desired state to the absolute 7B0F address locally, i.e. passing the information over the network is not required. Therefore the volume of network traffic is halved in a CAMP based system and will improve the efficiency of CSMA/CD type LAN in a FMS.

Furthermore, information exchange between entities in a common-memory system requires the obtaining of the base address for the desired entity in the common memory. As the number of cell controllers is increased in the system, problems will arise for allocation of new memories in the common-memory. However, in both types of systems reading of the information from another entity (cell controller) requires utilisation of the network. From this it is evident that the distribution of the common-memory among entities will provide the FMS with several advantages and this is the way in which entities in the Durham FMS hold their system status.

Each entity in the FMS polls its CAMP to find out whether it is invoked for a particular service by the other entities in the network. The method of polling is adopted since each entity in the FMS is involved in the real time operation of programmable devices, which must not be interrupted. Therefore, a policy is encorporated which polls the network following the execution of the current task, to look for any requests by other entities.

Entities in the FMS can get a snapshot of the activities and status of various devices with the aid of CellTalk. For example, a cell controller can get the status of a single device, a number of devices, or all of the devices and activities of any other cells. The way in which this is performed is that the CellTalk of the cell controller carries out a direct memory access of the CAMP of the distant cell controller before providing the information to the local cell controller. This is one of the important features of CellTalk, as it can be utilised as a monitoring tool of different entities for the user. A program namely, "MONITOR", Appendix F, was developed to demonstrate this. A user can monitor the activities of any of the entities in the FMS by only entering the name or the logical address of the desired entity. This program was found to be very useful during the debugging and testing of software modules and, due to the nature of the system software, tasks being processed in distributed processors concurrently without it would have been extremely difficult.

CAMP holds information in a convention similar to that laid down by the Manufacturing Message Service (MMS) of MAP. This allows easy interface of the FMS to the MAP network, since the information within the FMS is handled in a similar way and uses comparable syntax to MMS. In this way CellTalk can offer the same functionality as that of MMS. The MMS standard was used to provide the basis for the selection of functions implemented here. Hence, if a MAP network is connected to the FMS, and for example, the status of a device is required by the MAP compatible entity, the CAMP, with the aid of CellTalk, will provide the entity with information in a MAP-like syntax and format.

As it is not necessary or desirable for most systems to implement the complete MMS service the following services were chosen as adequate for the Durham FMS: Up-load and Down-load to CNC from computers; Dynamically down-load programs to CNCs ; Report change of status from NCs; Start and Stop program execution in NCs; Request status information from NCs; Transfer files between computers.

The network communication protocol developed within the CellTalk, Appendix F, functions according to a "client-server" model shown in Figure (5-10). CellTalk resides within the entities of both master and cell level, and it allows assumption of position of either a client or a server. CellTalk initiates the communication process between the client and server by first establishing a logical association between the two parties. The client provides the server with information such as its logical address in the network (i.e. node identifier) and the channel on which it is transmitting. Once the server receives a request for an association, it sends a response to the client to confirm its readiness. The client can then either send messages or control commands, transfer files to the server, or request a service from it. During a data transaction between the client and server, the party which is sending the data will perform a combination of software and hardware checking to ensure whether or not the data sent is being corrupted. Should a data frame get damaged during the transmission process, the transmitting party will retransmit the same data. Hence CellTalk provides a measure of fault tolerance in the noisy manufacturing environment.

In order to make the process of utilisation and development of communication routine as simple and as user friendly as possible, a hierarchical structure for information exchange was introduced, Fig. (5-11). As this made it possible for the lower level services and primitives to be transparent to the user.

5.8 Communication Primitives

All the entities in the FMS which take part in the process of communicating with one another have a series of routines enabling them to exchange information. These routines are referred to as communication primitives. Figure (5-11) shows the hierarchy of information exchange for the FMS. All information on the network is sent in packets or frames and these are assembled and disassembled by an ADLC chip [Acorn Econet 1983]. At the lowest level in the communication hierarchy, during a single "data transmit" operation, four frames are exchanged. This is known as a four-way handshake and works as follows;

- the initiator entity sends out an "are you there?" frame, called a scout,
- the responder entity sends back an acknowledgement,
- the initiator sends its data,
- the responder returns a final acknowledgement.

At the next level up, direct memory access takes place by two procedures namely, PROCpeek and PROCpoke. These procedures allow information to be read from or written to the CAMP of a remote entity respectively. At this level there exists an error checking routine which can identify the nature of communication hardware problems during a read or write session. Transfer of messages between entities is also carried out at this level and is performed by PROCkeyboard-poke.

At the highest level of file transfer, information exchange and mailing are carried out. Routines at this level use the services offered by lower levels which are transparent to the user. For example, if at some point during the process of part production the scheduler wanted to order a cell controller to end the process of part production and reset devices in that cell, all that is required of the scheduler is that it sends a series of messages for the devices to be stopped and reset. This is done by utilising procedure PROCtx-message. This procedure in turn makes use of services at the lower levels (i.e. PROCpeek, PROCpoke etc.). At the cell controller the message is received by PROCget-mail before being tested by PROCmail-test to obtain the type and nature of the message. Once the message is recognised, the appropriate control command is then issued to the devices involved by the cell controller.

`CellTalk` can perform file transfer in two modes. In mode one the transmitting party sends the length of file to be transferred to the recipient party, before sending its content byte by byte. In mode two, the transmitting party sends the name of the file to be transferred and the other party itself loads the file from a network common data storage (i.e. file server). This is a more efficient way of transferring files from one entity to another, but it relies entirely on the presence of a file server or network common data storage.

Sending messages from one entity to another can also take place in two modes. In the first mode the message length has to be specified before transmission of the message. In this mode there is no limit to the message length. However, in the second mode there is no need to specify the message length but there is a limit on the size of message transmitted (255 bytes) at a time.

`CellTalk` can perform Remote Procedure Calls (RPC). This commences once a client establishes a logical association with a server. The client then sends a request for a particular service from the server on the network and this is performed by `PROCinfo-request`. After the server has received the request it processes it and acts upon it accordingly. The server will then send the computed result to the client and this is done by `PROCrespond`. Application of RPC can be demonstrated when a cell controller (client) requests the part orientation from the previous cell (server), and the server, after obtaining and computing the part orientation, provides the client with the X and Y coordinates, together with an angle which is referenced to the datum. This idea can be taken further where the cell controllers would negotiate with each other and arrive at a decision on the basis of mutual agreement among themselves on how to process a part. In this way decision making at the cell level is decentralised considerably and the global information is reduced by locating decision making where the data originates.

5.9 Conclusion

A communication module was developed which enabled rapid and direct information exchange among all of the entities within a FMS. This module was implemented on the experimental FMS at Durham which possessed a baseband LAN, namely Econet. An attempt was made to produce a communication technique to be network independent and Econet was chosen to produce a working laboratory system. Common Address Memory in distributed Processors, (CAMP), was used to provide the system with the capability for distribution of information and decision making. CAMP was also used as a dynamic database, holding the system status data and being updated rapidly. CellTalk together with CAMP enabled different cell controllers to exchange information freely and directly leading to the term "heterarchical" control.

Since each cell controller made use of its own processing capability and updated the cell states within the cell's CAMP without passing the information over the network, the volume of network traffic was reduced considerably. The idea of CellTalk can be taken further to be employed in environments which require integration and intercommunication of non-standard intelligent devices.

Although CellTalk is a communication technique and it is hardware and operating system independent, easy implementation of this technique to the Econet resulted in making evident one advantage of Econet over other types of networks. This advantage lies in the easy memory access of entities in the system and

CellTalk took full advantage of such a facility. However, other networks may not allow direct access to the memory of entities in the system. Therefore, the following method could be employed instead.

Each entity would use the Remote Procedure Call (RPC) to obtain information from other entities. CAMP could then reside in a separate RAM interfaced to each entity (cell controller) input/output facility (i.e. I/O RAM). The size of this RAM is very small as 1K of RAM can hold 1024 separate system/cell specific data with each memory capable of holding 256 different states. Each cell controller will then, instead of storing various states into memory locations (in CAMP), encode the data in the form of a binary signal (bit pattern) before outputting it to the I/O ports where this data is then written to the I/O RAM. A cell controller could use a RPC to obtain specific data from another cell as the RPC allows the entities of a cell controller to make procedure calls on other cell controllers across the network. For a cell controller to obtain specific data on another cell it first calls a procedure to send a request to the distant controller. Upon receipt of the request, the distant cell controller will read the content of the desired memory location (one location assigned for each cell specific data) in the I/O RAM. Once the distant cell controller has obtained the requested information it will send a reply to the requesting cell controller. Since all systems have some kind of input/output capability the developed software and I/O RAM could then function in any system hardware or operating system.

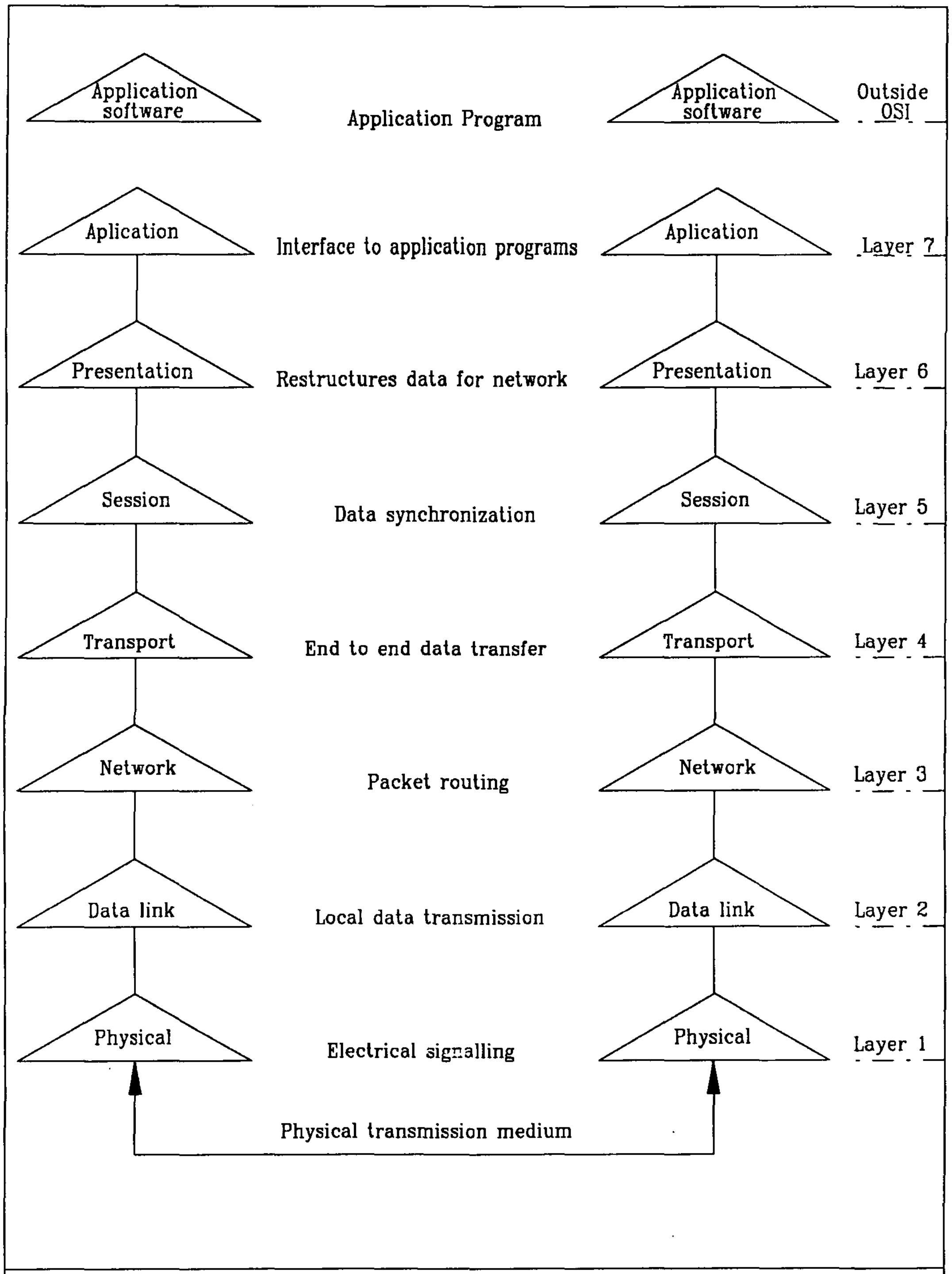


Figure (5-1) OSI reference model.

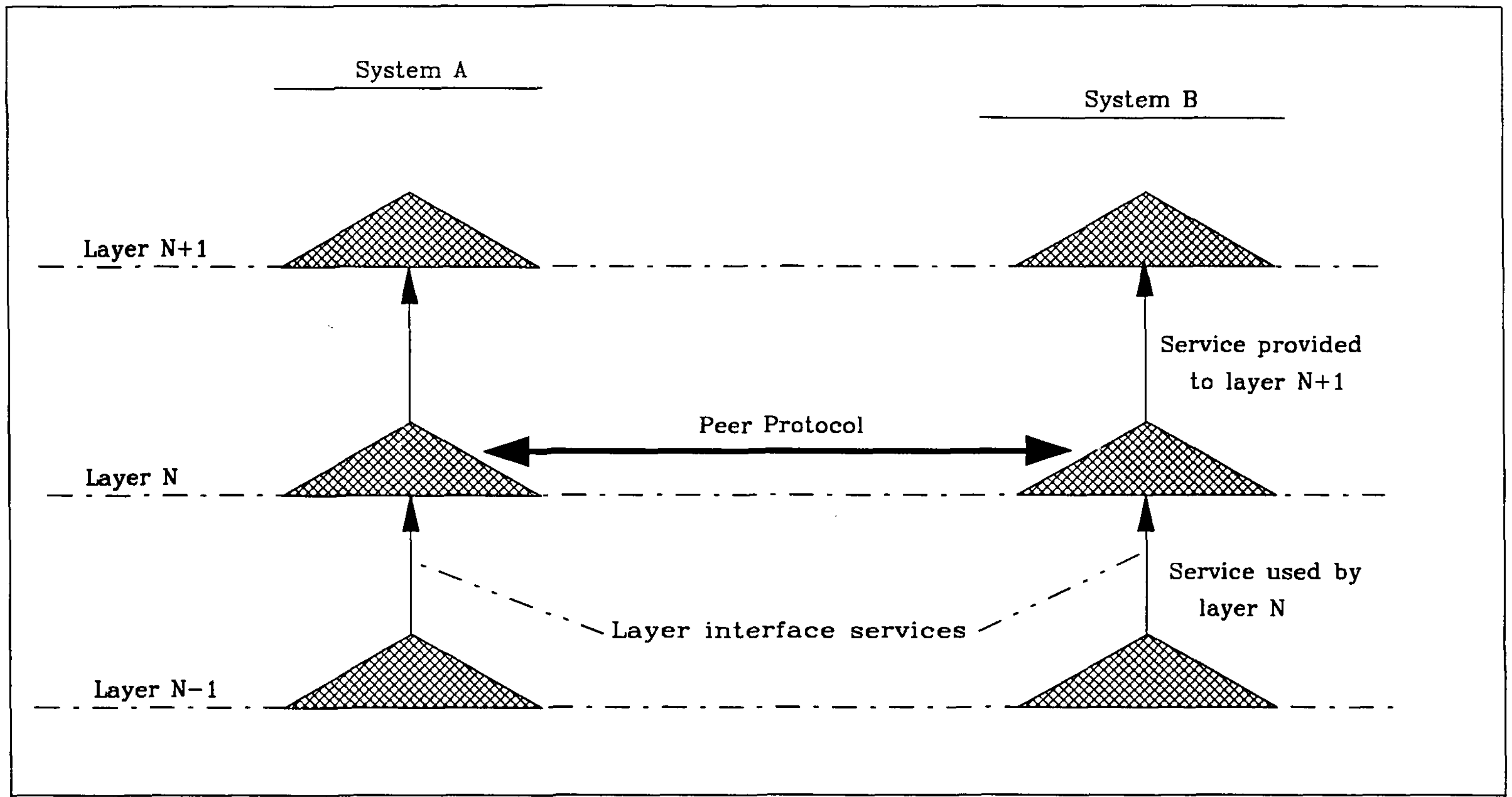


Figure (5-2) The interrelationship of a layer entity.

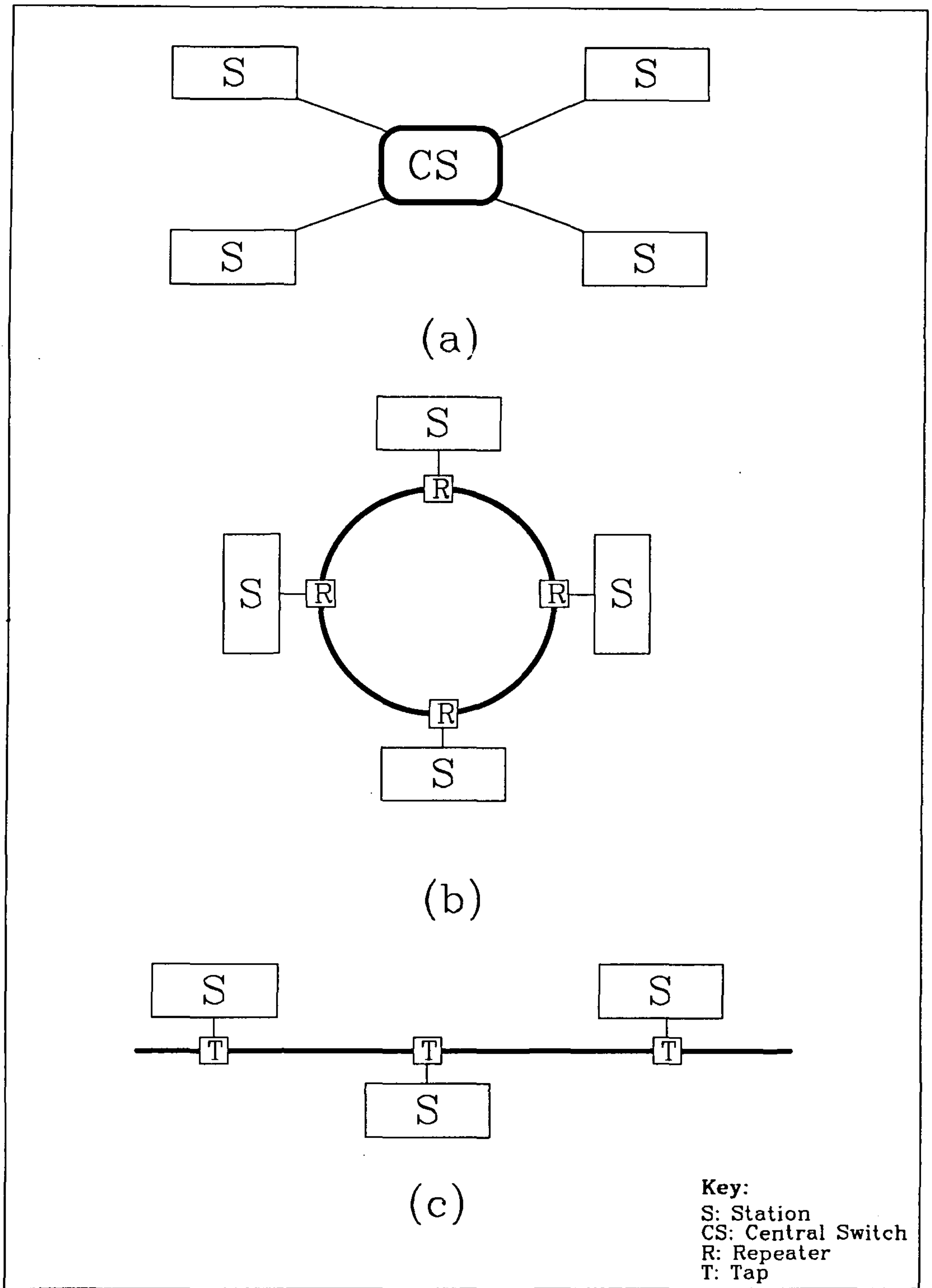


Figure (5-3) Network topology, (a) Star, (b) Ring, (c) Bus.

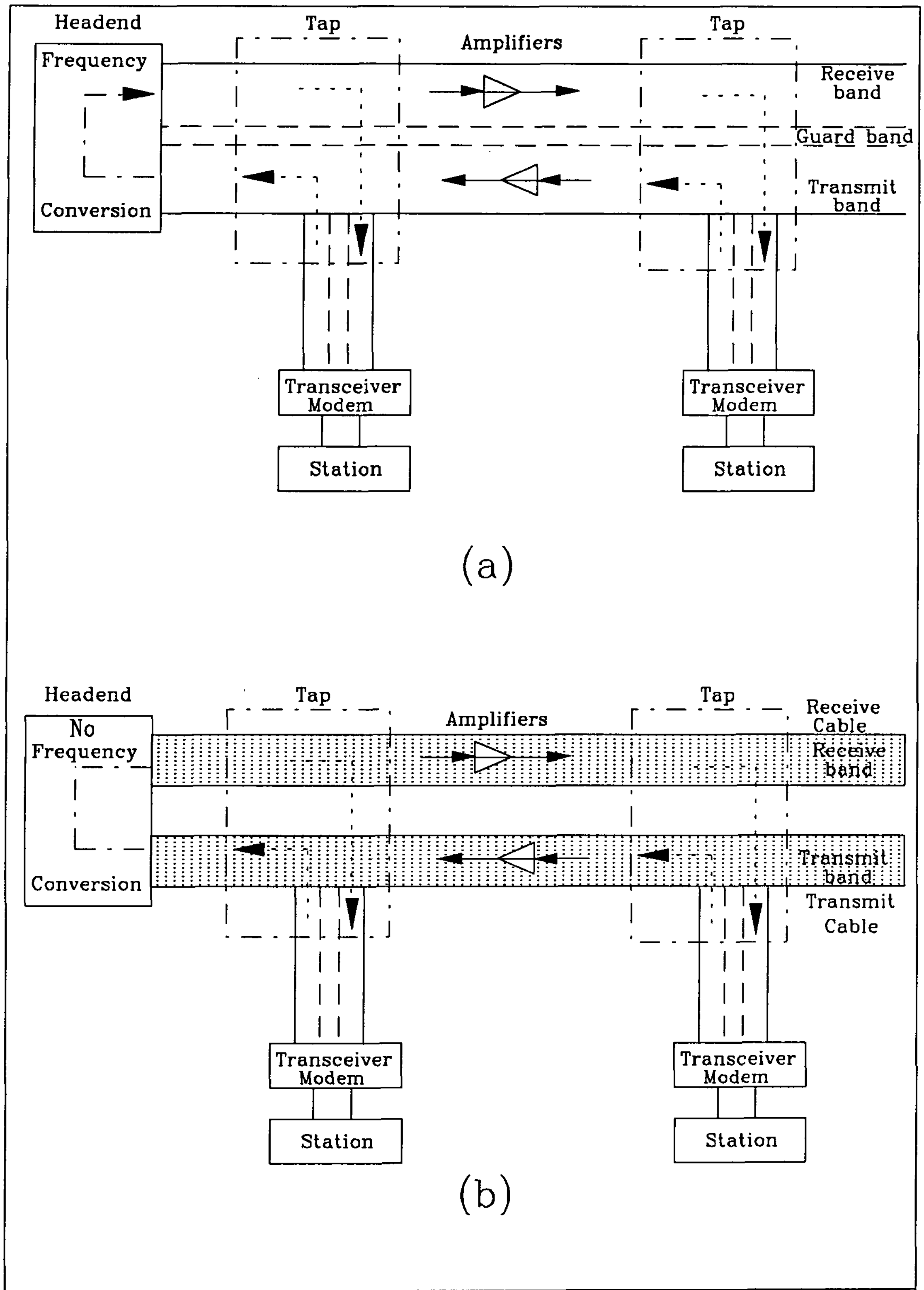


Figure (5–4) Signalling techniques on broadband with (a) Single cable midsplit, (b) Dual cable.

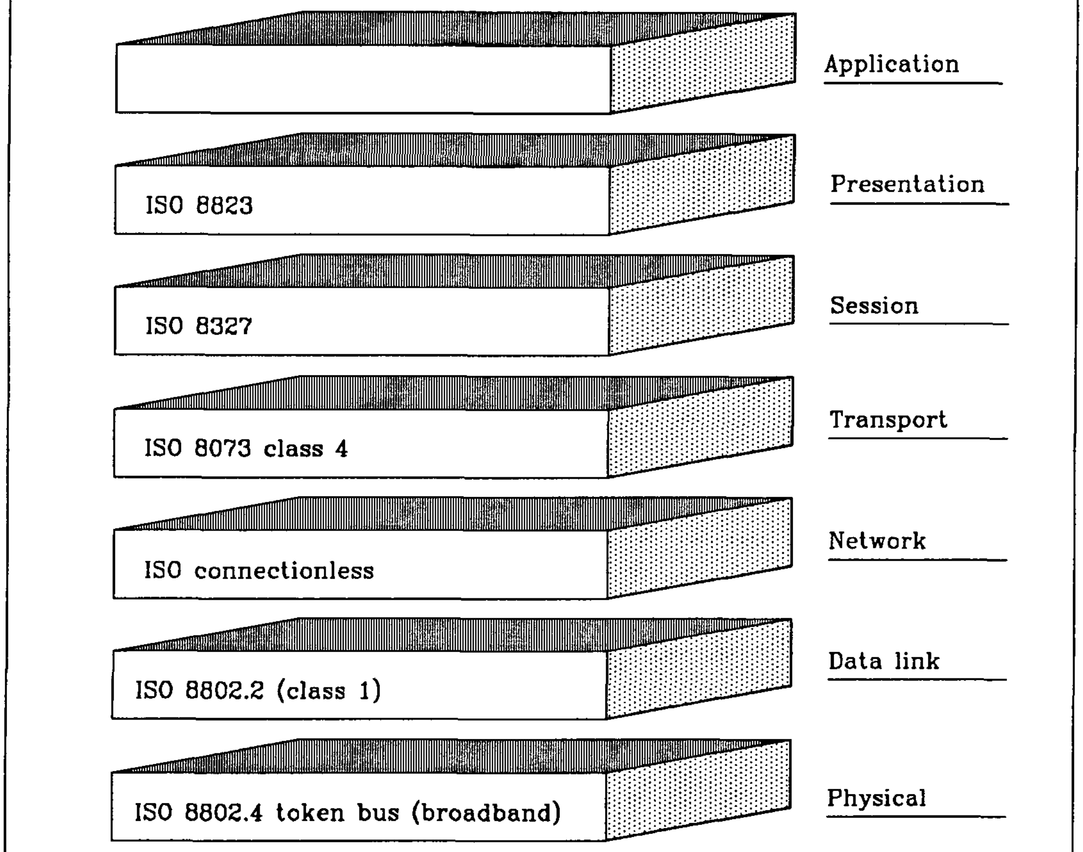
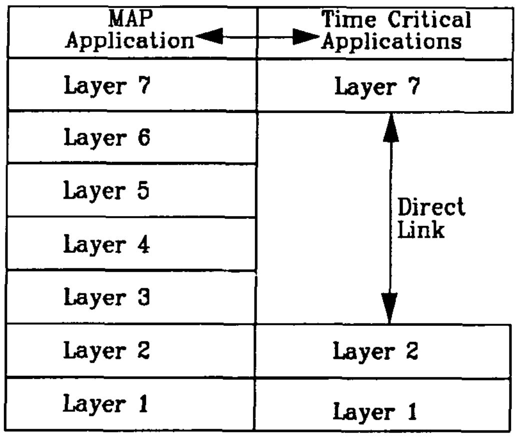
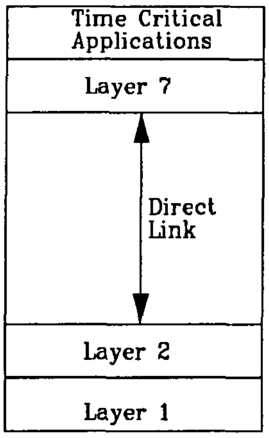


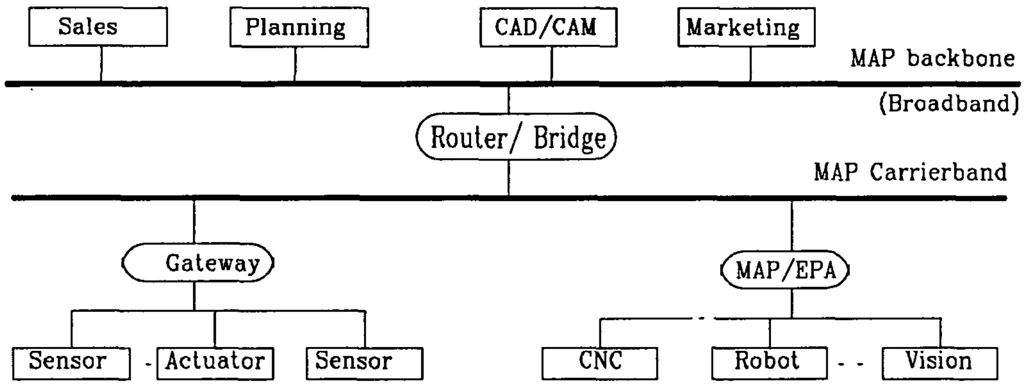
Figure (5–5) MAP 3.0 layer protocols.



(a)

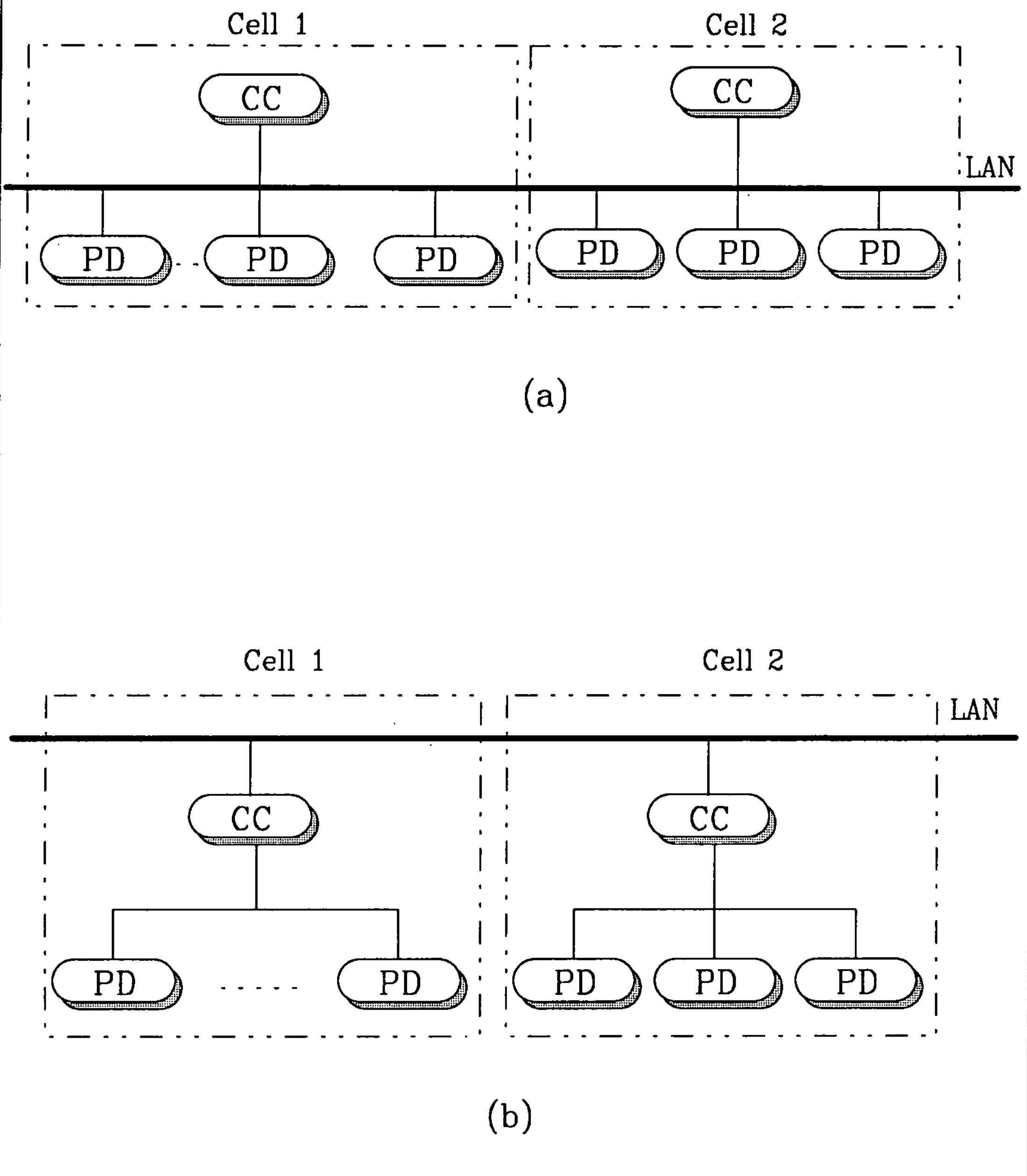


(b)



(c)

Figure (5–6) MAP networks; (a) MAP/EPA, (b) MiniMAP, (c) A typical MAP network.



Key:
CC: Cell Controller
PD: Programmable Device

Figure (5-7) Arrangements for FMS controllers;
(a) requires deterministic
(b) requires non-deterministic network.

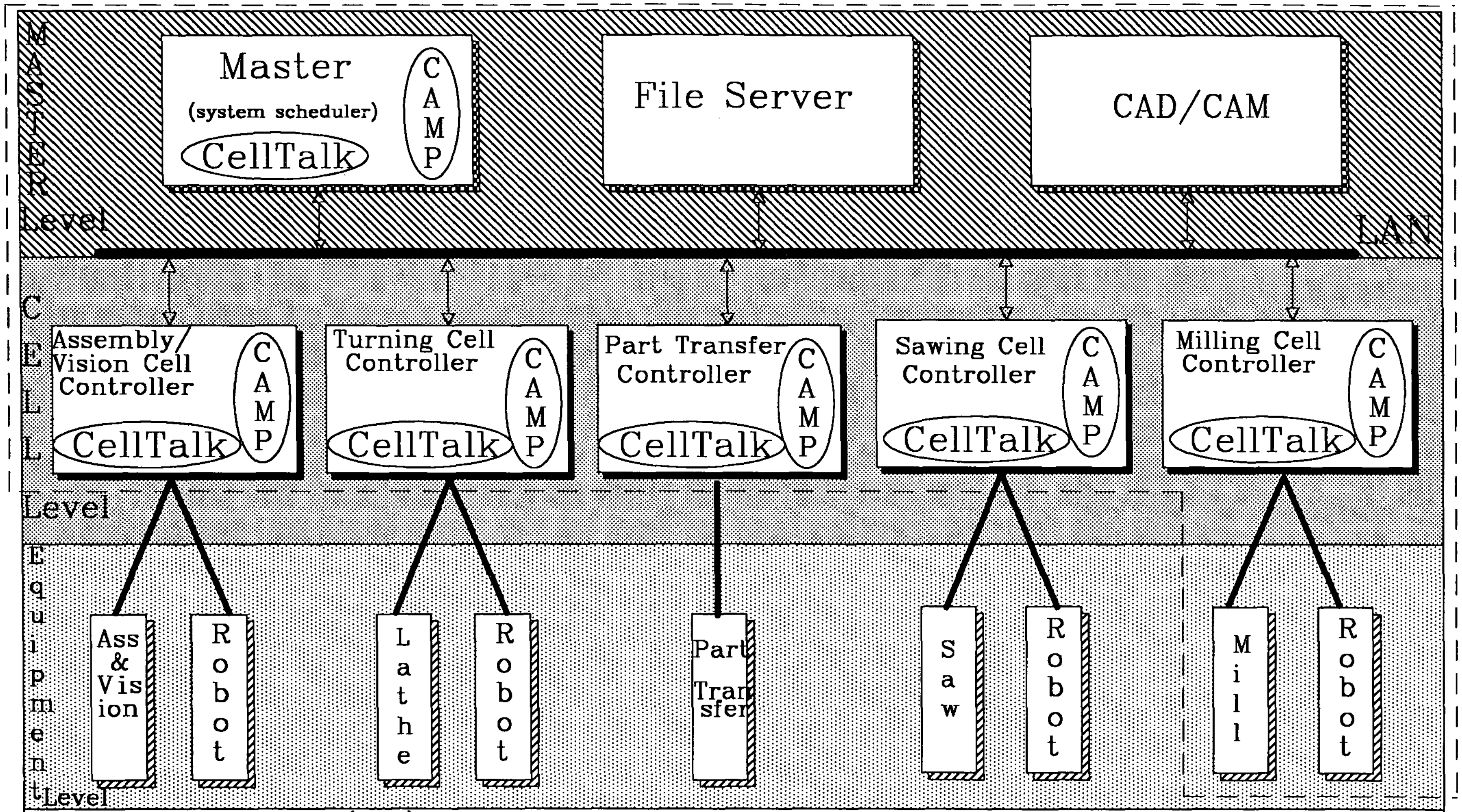


Figure (5–8) The logical position of CAMP and CellTalk in the Durham FMS.

Remote Procedure Calls may take place between any two cell controllers and between the master controller and any other controller by means of the LAN.

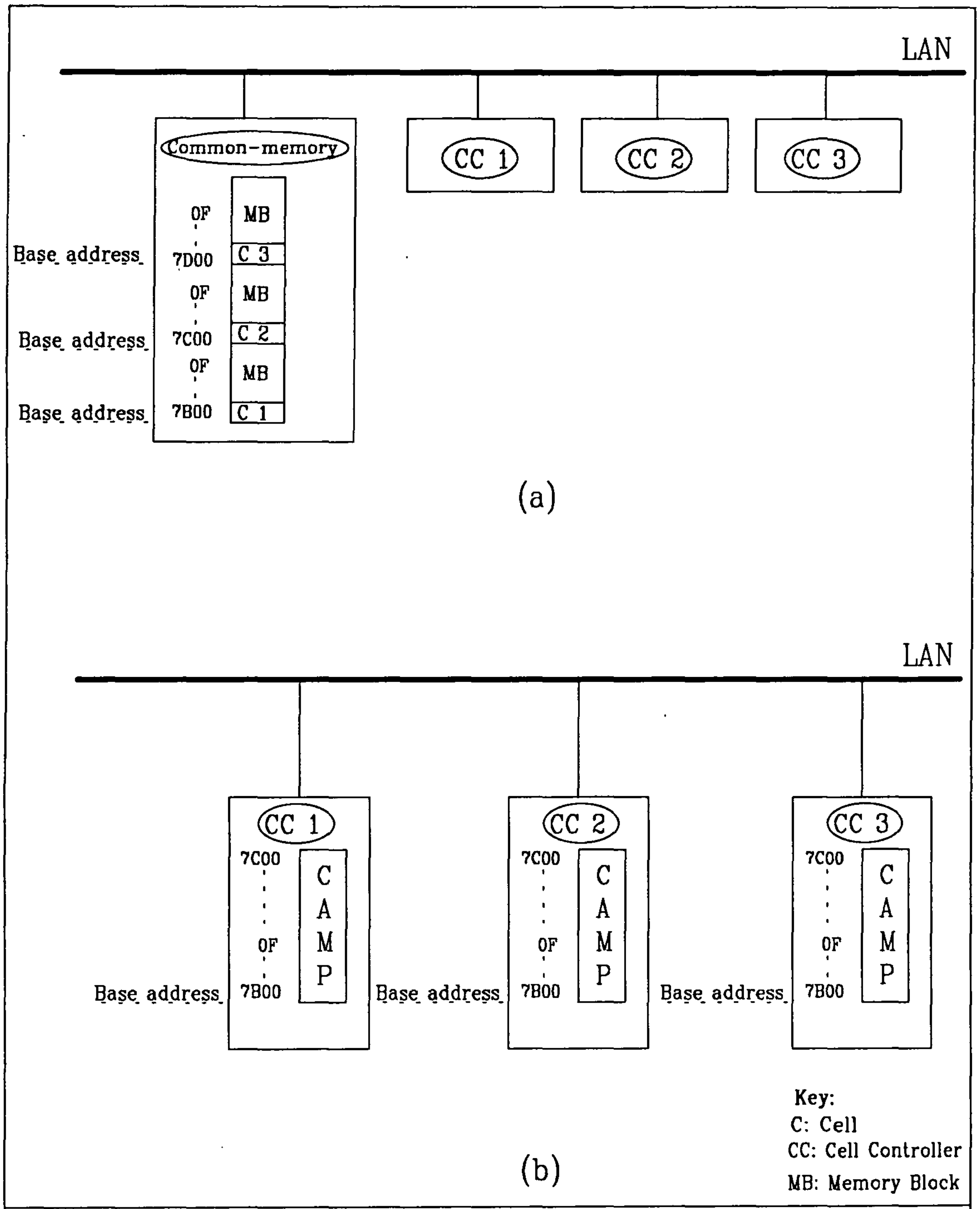


Figure (5–9) Reserved memory for the cell specific data;
(a) Common-memory based system
(b) CAMP based system.

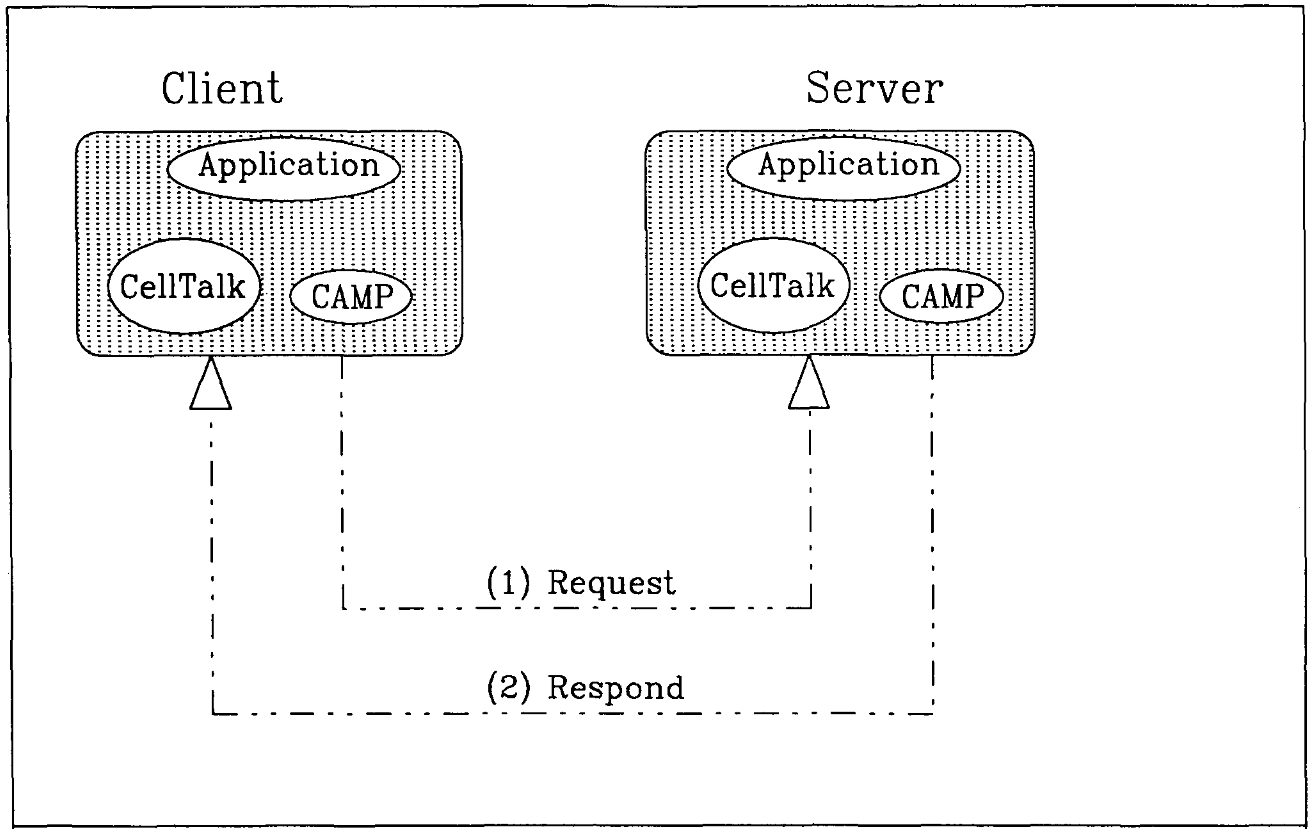


Figure (5–10) Data exchange between client and server entities.

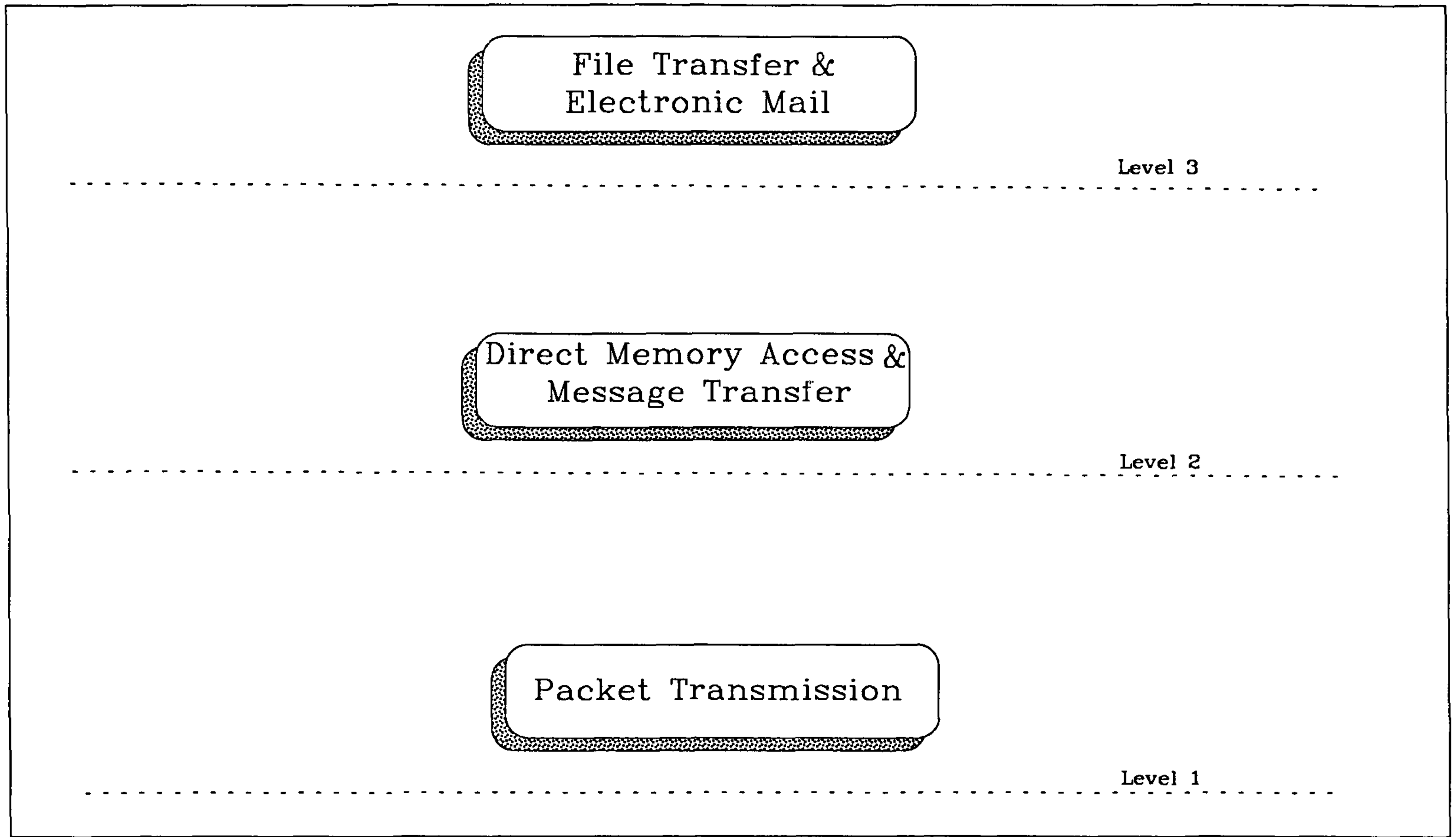

File Transfer & Electronic Mail
Level 3
Direct Memory Access & Message Transfer
Level 2
Packet Transmission
Level 1

Figure (5-11) Hierarchy of information exchange in the CellTalk.

Table (5-1) Common Addressed Memory in distribute d Processors (CAMP).

Address (Hex)	*Address name*	*Set by (Write)*	*Read by*	*Memory content description (Hex)*
7B00	logi-assoc	MA& CC	CC	00=disconnect FF=Connect
7B01	client-st-no	MA	CC	station number
7B02	client-port	MA	CC	port (communication channel no.)
7B03	server-resp-mes	CC	MA& CC	00=not-ready FF=ready
7B04	mass-stat	CC	MA& CC	00=processed FF=processing 01=message-not-recognised
7B05	prog-stat	CC	MA& CC	01=down-load successful FF=program running 00=completed
7B06	in-partbuf	CC	MA& CC	number of parts in the input buffer
7B07	out-partbuf	CC	MA& CC	number of parts in the output buffer
7B08	production-period	MA& CC	MA& CC	00=completed FF=started 02=stopped
7B09	upload-prep	MA& CC	MA& CC	00=completed FF=in-progress
7B0A	part-trans	MA& CC	MA& CC	FF=part to be transferred to/from the cell 00=part transfer completed
7B0B	where-part-trans	MA	CC	01=part transfer from the trolley to the cell 02=part transfer from the cell to the trolley 09=default state
7B0C	server-resp-part	CC	MA& CC	00=not-ready for a part transfer FF=ready for a part transfer
7B0D	robot-stat	R	CC	00=free FF=busy
7B0E	trolley-stat	CC	MA	00=free FF=busy
7B0F	device-stat	CC	MA& CC	00=completed 01=idle 02=stopped 03=critical operation in progress 04=device reset 05=device inoperable 06=uncorrectable error detected 07=correctable error detected 08=diagnostic running
7B10	defect-part-stat	MA& CC	MA& CC	FF=part defective 09=part is not defective
7B11	nof-defect-part	CC	MA& CC	number of defective parts
7B12	total-nof-parts	CC	MA& CC	total number of parts in the cell
7B13	nof-processed-parts	CC	MA& CC	number of processed parts

Key:		
	CC	: Entity at Cell Controller level
	MA	: Entity at Master level
	Address	: Reserved memory locations address

CHAPTER SIX
System Information Collection and Representation

6.1 Introduction

In a Flexible Manufacturing System up-to-the-minute information is required to keep operations on track, to meet production goals and to make high quality products. By closely watching the manufacturing process through intelligent computer monitoring, problems that reduce production levels or production quality can be quickly identified for corrective action. Immediate feedback information on the shop floor is needed to provide data such as; which machines are down, how many parts have been through certain operations, what are current defective parts rates and are the finished parts meeting required quality standards. Collecting this information fast and reliably and making it available to the decision making modules (or human operator) when it is needed, can make a critical difference to equipment utilisation and product quality. As the available technical information is becoming more copious with short decision times, collecting, presenting and archiving of available data is absolutely essential to the comprehensive documentation of the production process [Scheer 1988].

Currently, one of the most useful ways of presenting the shop floor data is by using graphical representation. As a change in colour or shape, as opposed to text or numeric values, induces an instant reaction in the user to act quickly, it is a quick way of conveying information. Every real-time element of the display is updated to reflect system status changes as they occur and, since the user has the true real-time information, response to and anticipation of the problems could be far more effective when they happen. For the user to get a better idea of the dynamics of the shop floor, pictures which represent the plant's production entities could be animated to give a realistic representation of process operation. Window techniques are used to display information from several parts of the shop floor (plant) on the screen at the same time and also a detailed information and process overview could be shown simultaneously.

This chapter describes modules developed for the recording and presentation of the activities when have taken place in the Durham FMS. Section 6.2 explains the disadvantages of information presented in the form of messages at the master and cell level and how this has been solved. In order to represent the system activities information, it has been encoded in a compacted form before being recorded for off-line system analysis and this is discussed in section 6.3. The recorded information is then decoded and represented graphically to ease the understanding of the system and its activities during a part production period. This is explained in section 6.4 which is then followed by a description in section 6.5 of the information which could be obtained from the graphical presentation of the recorded information. Finally, section 6.6 describes how the up-load information on FMS cells is prepared by cell controllers for historical and archiving purposes and what its advantages are.

6.2 Real-time System Scheduling Report

The algorithm developed for the scheduler to coordinate the overall activities of the FMS provides messages on the part production progress, at master and cell levels, throughout the period of part production. These messages in effect are short reports on production activities and what the system and the cells have actually performed at that time. The report is given by the Monitoring Modules

(MM), residing at both master and cell levels at the end of every monitoring cycle, which is discussed in chapter four. The reports were found to be most useful for the debugging of the scheduler and cell controller softwares. This on-line report generation had to be in the form of text messages due to the lack of available memory in BBC microcomputers. The teletext mode of BBC is very economical in the use of memory and requires only 1K of RAM, unlike the graphic modes which may require 10-20K of the available 32K of memory to map the screen [Coll 1982], [Dickens 1987]. Furthermore, part of the high resolution graphics memory is used by the CAMP, discussed in chapter five.

Reading the text messages will give a comprehensive explanation of the activity at the time of its occurrence. However, it may also cause confusion over the understanding and identification of the stages of progression in part production of the system as a whole. Thus, the graphical representation of the system and scheduling activities was the only acceptable solution. It retains one drawback, however, that although the information regarding the scheduling activities has been recorded in real-time, due to the lack of available memory, as stated earlier, it had to be utilised off-line. This graphical representation is being named; "System-wide Scheduling Chart".

Had a restriction on the amount of available memory not existed, the scheduler would have had a routine to illustrate the overall activity of the system, graphically, in real-time, and to predict the oncoming activities as a series of simulated objects for the cells, machines, parts, etc. This routine could have been extended for the simulation of the system activities off-line. The information that this simulator could then provide would include:

- prediction of the total production time of a job (batch).

- indication of the best part production route.
- anticipation of the number of tool changes should a tool reach its life limit in processing of the desired batch size, etc.

In order to demonstrate the effect of extreme conditions on the part production of the system and the cooperation capability of various system software modules (SIT) developed for this project, the processing time of a part in different cells was varied, the logical route of part production was changed and the batch size, ten good parts to manufacture, was kept constant. A test number was given to each condition. For example, for a job with three active cells the processing time for each cell was set as short, an arbitrary value of 18 seconds, or long, an arbitrary value of 48 seconds or longer, in a combinational manner. Section 6.5 includes an analysis of jobs with equal processing time for the like cells but a different logical part production route.

During a period of part production all the events which have taken place in the system are recorded by the scheduler at the master level and since this information is recorded in real-time and will vary with each job and the conditions at the time of its processing it can be used for off-line analysis to study the behaviour of the system and enhancement of system efficiency.

6.3 On-line Recording of the System Events

At the beginning of the scheduler program a test number is required from the user. This test number forms an integral part of a file name onto which the scheduler's events regarding the job under processing is recorded. The scheduler's events file name is simply constructed by adding the following character string/variables together.

i. (`file name`); ii. (`-A`); iii. (`test number`).

For example if the name of a particular job is `JOB4` with a test number of 8, then the created file name is encoded as `JOB4-A8`. Note that, characters `-A` (activity) is used to identify the type of file in the file server directory.

As the scheduler scans through the production cells to identify the one in need of a part transfer to or from a cell, the following events are possible:

a. a part must be transferred from the warehouse to a cell;

b. a part must be taken from one cell and transferred to the next logical cell;

c. a defective part must be transferred to the finished parts store;

d. a good part must be transferred to the finished parts store.

The scheduler uses the computer's real time clock and is set to zero at the beginning of the period of part production. Whenever the scheduler performs a part transfer task to or from a cell, from the warehouse or to the finished parts store, it records the time at which it performed the task together with the information leading to the identification of the source and destination of that particular part.

Throughout the processes of part production each part may have to undergo one or more operations in different manufacturing cells in a particular sequence (order) and this order is user defined for a job. Each manufacturing cell then can have a unique sequence number which will represent the logical position of the manufacturing cell within the system for a specific job. The user will define this sequence number for each cell at the start of a job definition period. The sequence numbers will always start from one, and zero sequence number is reserved for the warehouse and finished parts store. During a batch processing period all parts are taken one by one from the warehouse and passed sequentially from one cell to the next/another cell (next logical cell), in the part production route and finally to the finished parts store. The scheduler encodes the system's event before recording it onto the scheduler events' file. Information is encoded so that the least disc space would be used to hold the data and this is done in the following manner. The cell's sequence number is followed by a character of "T" for 'to a cell' or "F" for 'from a cell' or "R" for 'raw part' or "D" for 'defective part' or "G" for 'good part' and followed by a number representing the time at which the process of part transfer took place.

Hence the string `0.00R2.32` represents a raw part taken from the warehouse when the time was 2.32 minutes past the starting time of part production, and string `2.00T6.37` represents a part which has been transferred to the logical cell 2 when the time was 6.37 minutes past the starting time of part production. The string `0.00D7.52` represents a defective part being transferred to the finished parts store at 7.25 minutes past the commencement of the production period and so on. In this way, as soon as a cell completes the operation on a part, the scheduler attends that cell and records the time at which it attended the cell. Note that this time may differ from the time at which the operation was completed on that cell, since there exists a possibility that two or more cells may require attendance by the scheduler at the same time, and the cells in need are attended to sequentially, one at a time, by the scheduler. A more accurate time can be obtained by allowing each cell controller to keep the operation completion time in its own entity and the scheduler then to invoke the cell controller for this value.

The actual encoding and the recording of scheduling events is done by the following procedures in the `SCHEDULER` program (Appendix F):

PROCpart-trans-from-warehouse;

PROCpart-trans-to-nextcell;

PROCpart-trans-to-warehouse;

PROCdefective-part-trans-to-warehouse.

Table (6-1) Encoding of the System Events

Encoded String	*Description*
10	Ten good parts required, i.e. batch size of ten
0.00R2.66	A raw part is taken from the warehouse at 2.66 minutes past the starting time of part production
1.00T2.70	The part is sent to cell one at 2.70 minutes past the starting time
0.00R2.76	A raw part is taken from the warehouse at 2.76 minutes past the starting time
1.00T2.80	The part is sent to cell one at 2.80 minutes past the starting time
1.00F3.29	A part is taken from logical cell one at 3.29 minutes
2.00T3.35	The part is sent to logical cell two at 3.35 minutes
2.00F4.62	A part is taken from logical cell two at 4.62 minutes
3.00T4.67	The part is sent to logical cell three at 4.67 minutes
1.00F4.78	A part is taken from logical cell one at 4.78 minutes
2.00T4.88	The part is sent to logical cell two at 4.88 minutes
3.00F5.17	A part is taken from logical cell three at 5.17 minutes
0.00D5.19	The part is defective and is stored under defective parts in the finished parts store at 5.19 minutes
2.00F5.50	A part is taken from logical cell two at 5.50 minutes
3.00T5.55	The part is sent to logical cell three at 5.55 minutes
3.00F5.84	A part is taken from logical cell three at 5.84 minutes
0.00G5.86	The part is good and is stored under good parts in the finished parts store at 5.86 minutes

Table (6-1) illustrates the recording of a typical encoded data for the part production of a job with three active cells and a batch size of ten. Adjacent to the encoded data string is a description of the corresponding events occurrance in absolute time, or minutes past the commencing time of part production.

The encoded information is useless to the user in its original form for the analysing of the system event unless it is first decoded before being represented in a graphical form for the user's off-line analysis.

6.4 Graphical Presentation of the System and Scheduler's Activities

A program, namely ACT-CHART, Appendix F, has been developed to demonstrate the recorded system scheduling events graphically, including the activities, defined below, of the scheduler during a part production period. This program resides at the master level and is executed as a result of the user selecting option number 9 in the main program (i.e. MASTER program) to view the graphical representation of the system scheduling activities for a job which has already been processed. Figure (6-1) illustrates a flow chart of the software. The algorithm requires a predefined job name and pre-run test number from the user. Next, it uses the manufacturing database to obtain the relevant information about the job. Once the software identifies the number and types of active cells which took part in the part production of the specific test number, it divides the computer's graphic screen into the same numbers of horizontal bands as the sum of the number of active cells plus two to represent the FMS facilities (one for the warehouse and the other for the finished parts store).

There are five graphic modes within the BBC microcomputer with various screen resolutions and number of characters per line. The different number of colours that these modes can offer vary according to the screen resolution. Since mode 0 has the highest resolution with the largest number of characters per line (80 characters), it was the most suitable mode in which to present the system-wide scheduling chart. This mode also offers two colours, which were utilised in the information presentation of the chart.

Figure (6-2) shows a screen dump of the system-wide scheduling chart for a job with three active cells. The area between two successive horizontal lines is representative of the character on the left-hand side of the screen, which is in turn defined at the top of the screen. For example, the area adjacent to the letter `M` represents the milling cell. The two successive parallel lines at the very bottom of the screen show the recorded activity conclusion time in minutes. Here an activity is defined as a part transfer either from the warehouse to the first logical cell, from one cell to the next logical cell, or from a cell to the finished parts store. This is depicted in solid arrow form.

Each part must take the logical route from the warehouse, close to the bottom of the screen, to the finished parts store, at the top. In doing so each part, with the exception of defective parts, will visit each adjacent cell (on the chart), positioned on the screen in ascending order of cell sequence number. Text messages to clarify the meaning of the character in use are printed at the top of the screen. Information regarding the job name, batch size, test number and the screen number are printed on the top right-hand corner of the screen.

Due to the fact that presentation of all of the scheduler's activities may not fit onto one screen, more than one screen may be required. The way in which this is achieved is as follows; a screen is set up each time by the procedure `PROCaxis` in `ACT-CHART` program and it is numbered in ascending order starting from one. Next, procedure `PROCplot-timing` will read information regarding the scheduler's activity and its conclusion time. This is depicted onto the screen and, when the screen is full, it waits for a carriage return, or the depressing of any key, by the user before the screen is refreshed, renumbered and ready for procedure `PROCplot-timing` to present the next list of scheduler's activities and their conclusion times. This process will continue until all the scheduler's activities are presented.

Dotted arrows or lines represent the trail of one activity to the next on the chart. The tail section of each solid arrow shows that a part is being taken from the cell, or warehouse, in which the tail section is present. This part is then transferred to the cell, or finished parts store, in which the head of the same solid arrow is present. There exists a dotted line or arrow after each part transfer and this can be interpreted as the duration without a part transfer. The reason behind this presentation of such dotted lines or arrows was to make the follow-up through the activities easier.

Part transfer between the cells, from the warehouse or to the finished parts store is carried out by a part transfer device, the pneumatic trolley. The solid and dotted arrows also illustrate the movement of this device, when carrying a part in solid arrow, and dotted when carrying no parts.

To follow the progress of activities; it always begins at the tail of an arrow, moves to the head, then to the right, or in the positive direction of the activity conclusion time axis, until the tail section of the next activity is reached. After the completion of a part transfer to a cell or finished parts store, the conclusion time of that activity is printed at the bottom of the screen. Since the printing of all of the activity conclusion times on a single axis would have been impossible, two axes parallel to each other at the bottom of the screen are utilised, and on each axis every other activity conclusion time is printed. Parts transferred to the finished parts store are labelled as "G" for 'Good' and "D" for 'Defective'. The sum of the good and defective parts should always equate the number of raw parts being taken from the warehouse.

At the beginning of the ACT-CHART program the user is questioned whether or not a hard copy of the system-wide scheduling chart is required. If the reply is affirmative, the program will perform a screen dump every time, after filling a screen with the activities information.

6.5 Obtainable Information from the System-wide Scheduling Chart

One of the most important features of the 'System-wide scheduling chart' is that it can demonstrate the various modules of the SIT, developed in this project, functioning correctly in order to achieve the manufacturing goals. An example is chosen to highlight the information, which could be extracted from the chart. Figure (6-2) illustrates the system-wide scheduling chart for a job with three active cells, manufacturing ten good parts, i.e. batch size of ten.

Throughout the system, each cell has two part buffers; input buffer and output buffer. In order to demonstrate that the system can cope with the minimal Work-In-Progress (WIP) storage area, and at the same time obtain the maximum utilisation from each cell, the size of each buffer is kept to a maximum of two. Thus, in this case, at all times the total number of parts in a cell, i.e. the total number of parts in the input part buffer, the output part buffer and on the machine's table should not exceed five. Parts transferred to a cell are placed onto the input part buffer and this information is shown by the Head of the solid arrow. In addition, parts are transferred out of a cell only from the output part buffer and the `Tail` of the solid arrow denotes this information. Note that the chart does not clearly display the exact number of parts in any part buffer at one time, and only exhibits the number of parts sent to or taken from a cell. Information such as the part processing times, part buffer sizes and part buffer status etc. of a cell can be obtained by constructing a similar chart (i.e. `Intra-cell Scheduling Chart`) for each cell from recorded cell data. Then each cell controller may be responsible for the recording of the In-cell activities. Currently, since each cell controller has a limited amount of memory in their present form, they are unable to cater for such tasks.

Conduction of simple checks on the system-wide scheduling chart can lead to the determination of the number of parts being processed in a cell. This is carried out by counting the number of tails of solid arrows. The number of parts sent to a cell should always be equal to the number of parts transferred out of that cell. This information is obtained from the chart by comparing the total number of the heads and tails of solid arrows in a cell.

Since all of the parts, except the defectives, for a given batch size must visit all of the cells, it can be said that, the number of parts sent to or taken from any cell must not be less than the number of good parts in the finished parts store. Furthermore, the total number of raw parts transferred out of the warehouse should always equate the total number of good parts plus defective parts in the finished parts store. The chart clearly illustrates this.

Assuming that the time taken to transfer a part from one cell to another is much less than the processing time of a part in any cell, then the time taken for n number of parts to be manufactured, where n is equal or less than the total number of parts in a batch, can be directly obtained by simply counting the number of good parts in the finished parts store; on reaching the desired value, the conclusion time of that activity, printed on the activity conclusion time axis of the chart, will indicate the total time taken to manufacture n parts. Hence, the total time taken to process a batch is the last number on the activity conclusion time axis.

The logical route of part production, is denoted by the characters on the left-hand side of the chart. All parts, except defectives, must visit the adjacent logical cell in the path of the part production route. This is shown by the upward solid arrows on the chart. A quick glance at the chart could provide such information.

A change in the logical route of part production may result in the production of a different number of parts, not less than the number of good parts required, in each cell, Fig. (6-3). The reason behind this would be the presence of a defective part along the logical route of part production. The assumption here is made that defective parts cannot be reworked (salvaged), and are not permitted to visit the next logical cell. In order to clarify the point, Figures (6-2) and (6-3) show two jobs, identical but for one difference, that of the logical route of part production. In both jobs the number of good parts manufactured are ten and the processing time of parts in the identical cells of both jobs are equal. The chart shows that the time taken to manufacture n number of parts, where n is less than or equal to the number of good parts, differs.

The chart demonstrates that the scheduler is functioning in real-time and that decisions are made on the basis of on-line feedback from the system. For example, should a cell produce a defective part, the scheduler will not send that part to the next logical cell. It also proved that the algorithm developed for the scheduler worked successfully for the part production of any size batch. Figure

(6-4) illustrates the chart for a job with two active cells and a batch size of three. Note the occurrence of a defective part or parts in each cell. Should the scheduling algorithm be extended to cater for like cells, the chart can then also show whether or not the load is being balanced between the cells.

The system-wide activity chart can be used to exhibit the movement of parts and the pneumatic trolley or AGV and the exact time of their attendance in a cell, warehouse or finished parts store. By fixing the size of input and output buffers, checks can be done on the chart to find out whether or not parts have been sent to a cell with a full input part buffer. This is achieved by counting the number of parts sent to a cell and subtracting the number of parts taken away from the cell. The input and output buffer size being two, the total number of parts in a cell must never exceed five.

This method of on-line recording of the system's events and the off-line chart utilisation and analysis can be employed for intra-cell scheduling (or sequencing). In doing so, the exact processing time of each part in every cell, the effect of tool wear on the part processing time, and the entry and exit time of all parts to and from a cell can be deduced.

In addition, an indication of the time that a part has been waiting, after being processed in the cell, to be transferred to the next logical cell would be available from the intra-cell scheduling chart. Assuming that the time taken for a part to be attended to by the system scheduler is very short, then the aforesaid waiting of a part in a cell would be due to the restricted size of the input part buffer of the next logical cell. Hence the optimum value for the buffer size of each cell can be obtained. Note that such optimisation on the buffer size of the cells is only beneficial if each cell is equipped with a part recognition module, such as a bar-code reader, if bar-coding of parts is employed, and a scheduler capable of processing multi jobs at the same time. Should a cell break down the exact time of the failure is recorded.

Although the System-wide scheduling chart gives information on the overall activities which have taken place in the system, it does not provide cell specific information for archiving purposes. A separate module has been developed for this, to function by the cooperation of cell controllers and the master program.

6.6 Up-load Information on FMS cells

This module was developed to show the capability of cell controllers for producing reports about the cell specific information and their performance and also, collection of these reports by the master at the end of a period of part production. During such a period the overall control of the system is performed by the entities at the master level (e.g. the scheduler). The scheduling module will pass the control to the main program (i.e. `MASTER` program) once it has learnt that all the cells have completed the part processing tasks (i.e. the batch has been processed successfully).

Messages are then sent to all of the active cell controllers by the master program to up-load a report on their cell specific information, such as cell performances after the production of a current batch. Each cell controller will then start computing and prepare and format the information before arranging with the master for the up-loading of this information. Cell controllers utilise the communication module for exchanging information with the master program. Presently, information such as the number of processed parts, the number of defective parts, the defect rate of the cell and the cell efficiency is computed before being formatted and up-loaded to the master. In this way the information processing takes place where the data has originated and this will reduce the volume of information to be processed by the entities at the master level. There are numerous calculations and information which are cell specific and local processing has shown several advantages, including the requirement for smaller memory and processing capability at the master level. At the moment, only a very small proportion of cell specific information is up-loaded. This is only in order to show the way in which the idea could be employed and further enhancement of this module may include other processing of additional information such as the amount of tool wear for this batch, remaining tool life etc.

The system allows for the existence of one up-load file per job and one record per each batch number of the same job. The `MASTER` program will search for the up-load file of the job which has currently been processed and, if the search is unsuccessful, an up-load file will be created for that job. Every time a job is processed, a record will be added to this up-load file to include the report on the `FMS` cells for the current batch. A batch number is given to a job every time the job is processed which denotes its order of processing among the previous batches of the same job. For example, a job with batch number 3 shows that this batch has been processed after two previous batches of the same job. Note that the batches may not necessarily have been processed consecutively. Preparation of the up-load file is performed by the `PROCprepare-upload` procedure in the master program.

Once the `MASTER` program has received a report from all of the active cells it rearranges the records of the up-load file in an ascending order of the number of processed parts (batch sizes) for easy understanding and tracking of processed batches. Since these batches were not physically produced but simulated by machines cutting fresh air, it did not seem important at that time to include the date and time of processing of batches. However, this information may prove vital in a real system, therefore including the time and date will be an improvement to this module. The software could then arrange the records of up-load file in ascending or descending order of time and date.

After a report on the FMS cells has been collected and recorded into the up-load file it will become part of the archived information which could provide historical information about the specific batch of a job. The archived or historical information can be obtained and viewed off-line at the master level. The user can achieve this by selecting option number 6 from the menu of the MASTER program. Once this option is selected it requires the name of the job. A search is then made in the database for the up-load file of the desired job which is followed by the exhibition of information on active cells for all of the batches processed to date for that job. The routine which performs this is called `PROChistorical-rep`.

Figure (6-5) shows the historical information of a job with three active cells. The software provides information such as; the total number of batches processed for this job so far, cell performance and cell specific information. It also computes the total number of parts which were required from the warehouse and how many good parts were produced for that batch and this is shown at end of the information for each batch. Cells, together with their up-load information are listed as such to present the route of the part production for the specific job (sawing, milling and turning in this case). As there is no cell with zero defect, each cell has required more number of parts than it has passed to the next logical cell.

One important aspect of historical reports on the FMS cells is that they can provide an input and some guideline for future FMS capacity planning. An improvement to this module would be the representation of this information in the form of graphical charts.

6.7 Conclusion

Different methods of information collection and representation were discussed. Although illustrating the system changes and status graphically is one of the best methods for obtaining the user quick response to the changes, for the Durham FMS, performing this in real-time was not possible due to the lack of available memory. Therefore, the system events were encoded before being recorded in real-time. This data was then decoded by a module which represented both the system and the events. The decoded data was represented in the form a chart which showed the system events took place during the period of part production for a specific job.

This chart enabled the user to study the behaviour of the system. The chart gave the cumulative time of the activities. In this way the user could obtain the time taken for a part to be manufactured for any number of batch size at a quick glance of the chart. The chart was also useful for diagnosing the occurrence of a system fault as the time and status of the system would have been recorded at the time of the fault. The chart was found to be very useful for the analysis of processed jobs.

This chapter also discussed how cell controllers up-load information to the entities at the master level so that the information could be recorded for historical and archiving purposes.

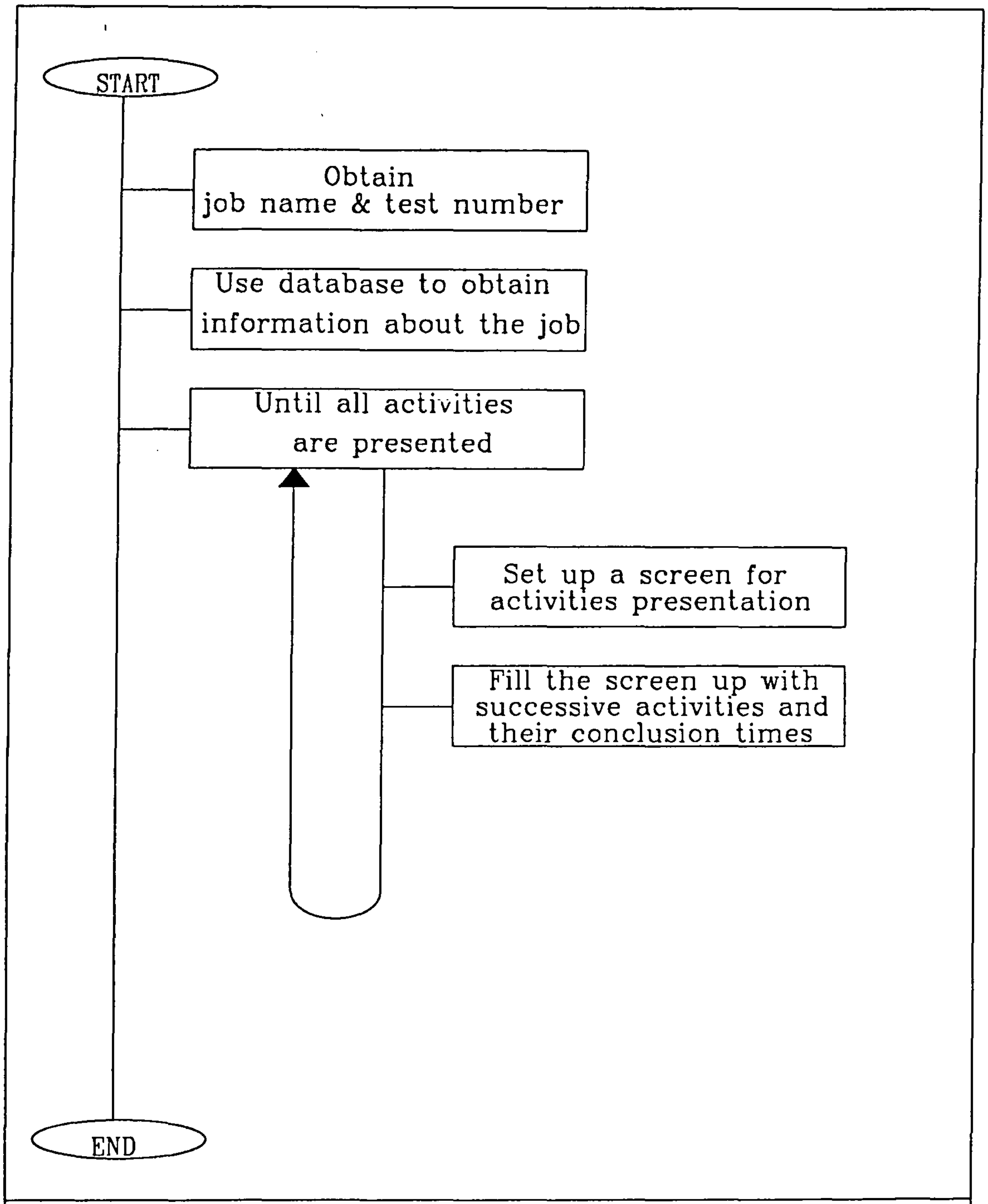

START
Obtain job name & test number
Use database to obtain information about the job
Until all activities are presented
Set up a screen for activities presentation
Fill the screen up with successive activities and their conclusion times
END

Figure (6-1) A flow chart for the System-wide activity chart program.

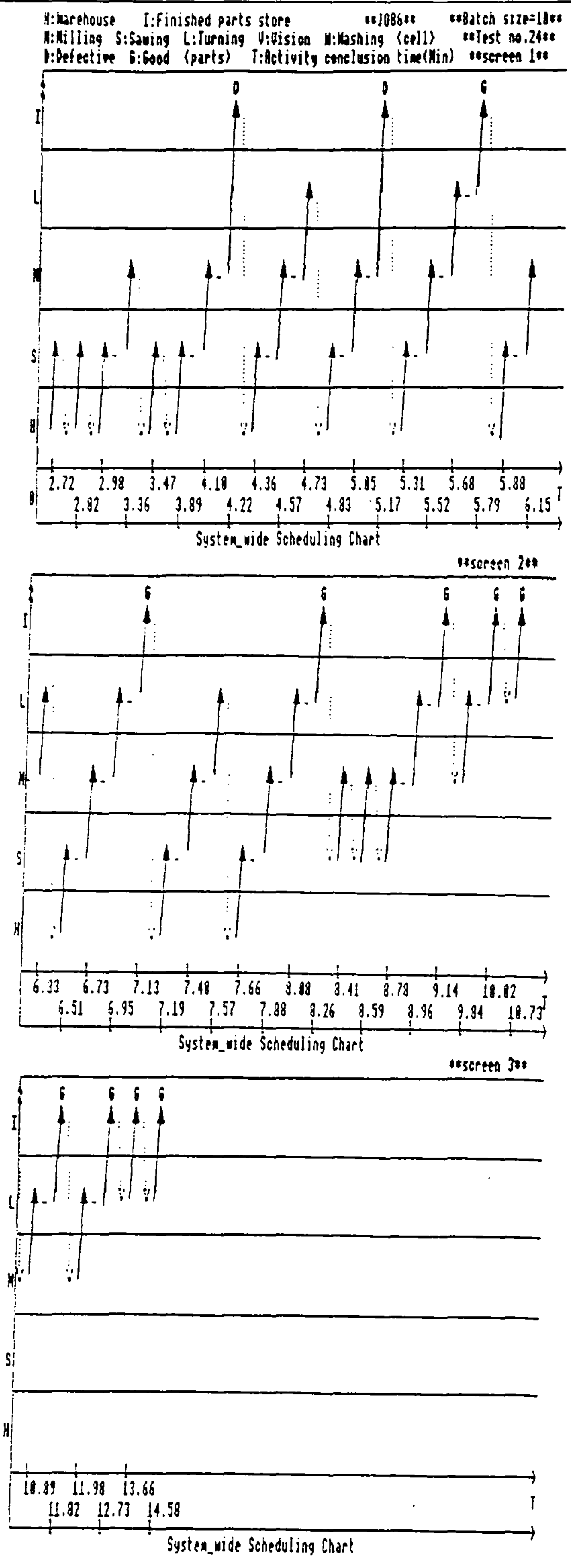


Figure (6–2) System–wide scheduling chart for JOB6.

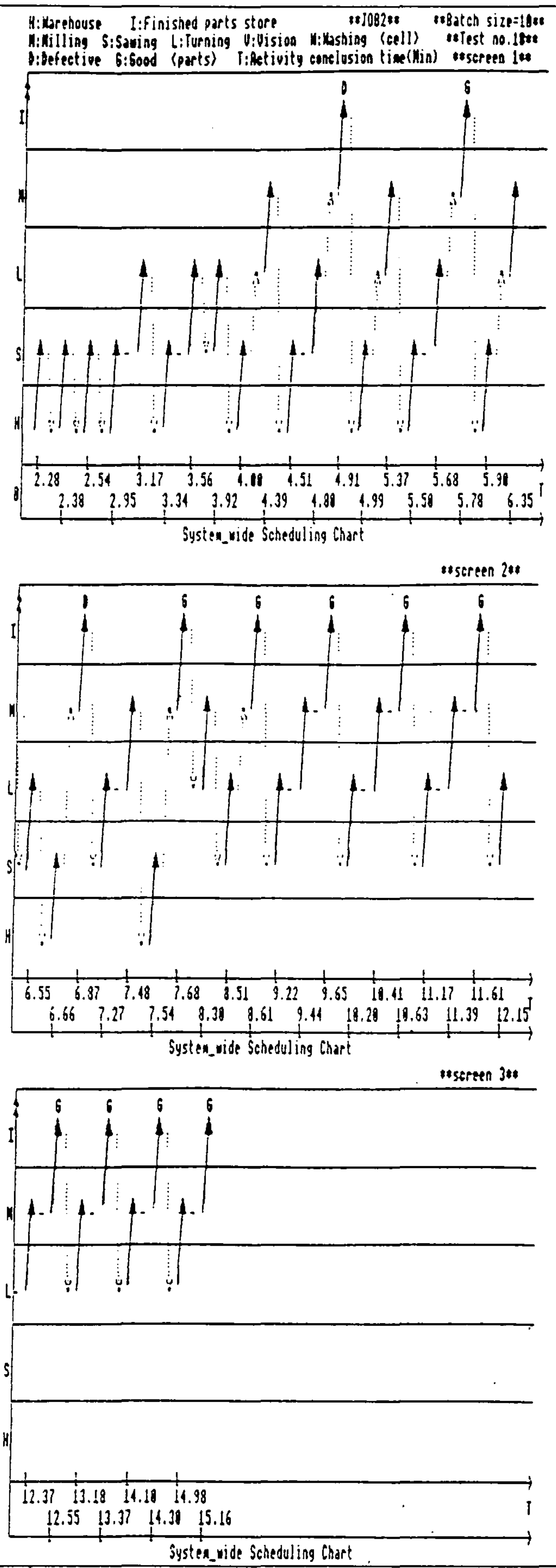


Figure (6-3) System-wide scheduling chart for JOB2.

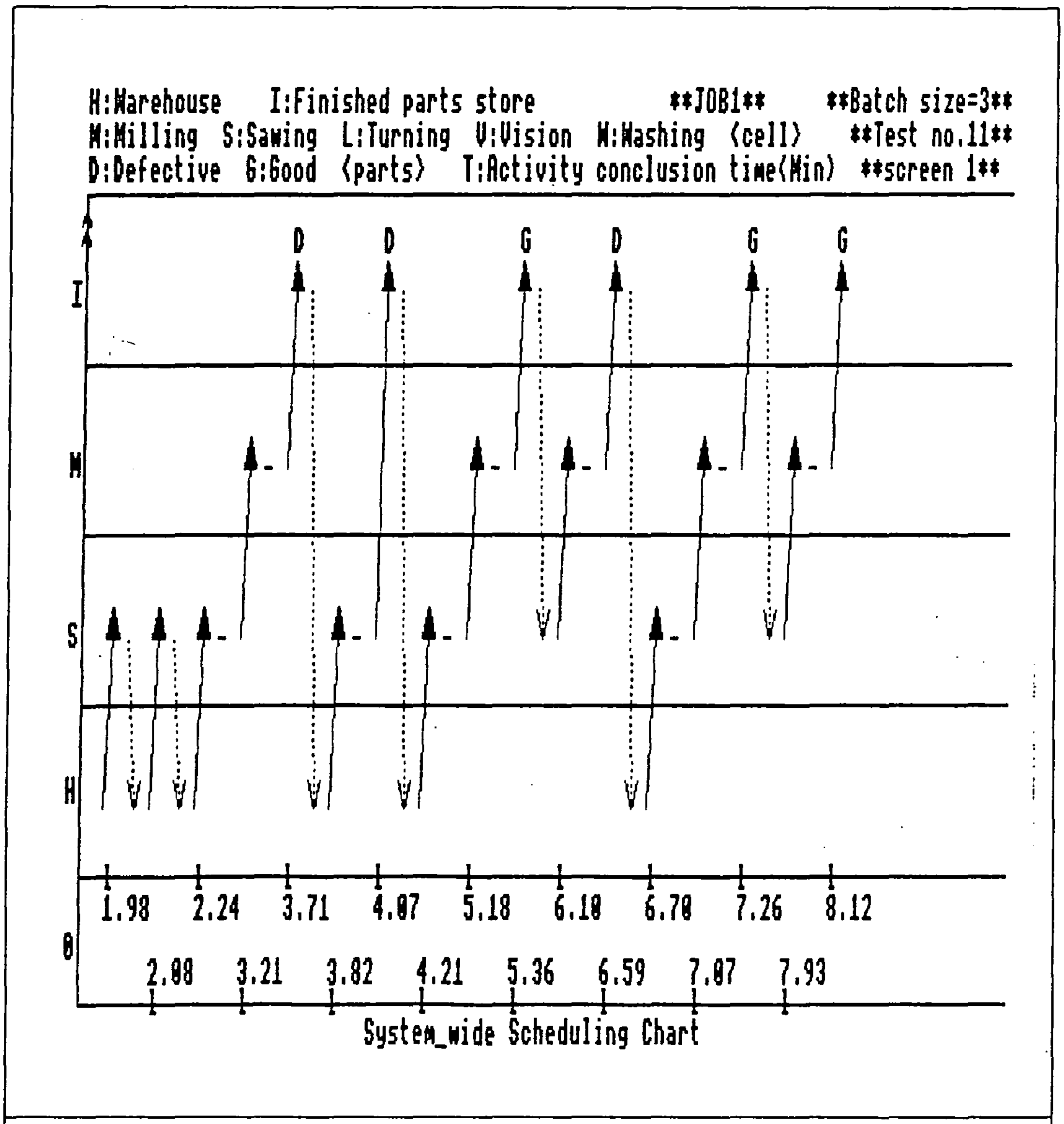
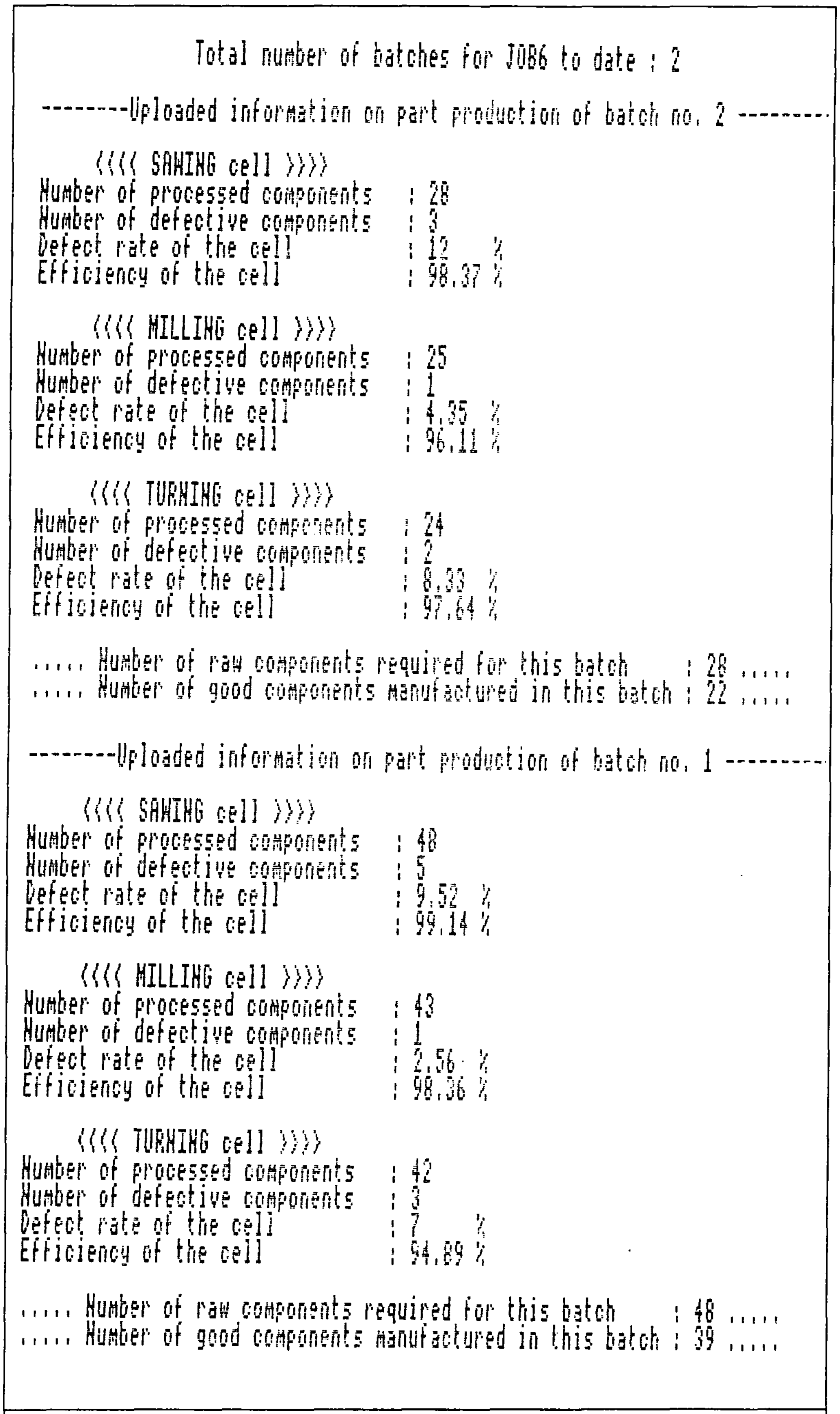


Figure (6-4) System-wide scheduling chart for JOB1.



```
                 Total number of batches for JOB6 to date : 2

 --------Uploaded information on part production of batch no. 2 ---------

      <<<< SAWING cell >>>>
 Number of processed components   : 28
 Number of defective components   : 3
 Defect rate of the cell          : 12    %
 Efficiency of the cell           : 98.37 %

      <<<< MILLING cell >>>>
 Number of processed components   : 25
 Number of defective components   : 1
 Defect rate of the cell          : 4.35  %
 Efficiency of the cell           : 96.11 %

      <<<< TURNING cell >>>>
 Number of processed components   : 24
 Number of defective components   : 2
 Defect rate of the cell          : 8.33  %
 Efficiency of the cell           : 97.64 %

 ..... Number of raw components required for this batch      : 28 .....
 ..... Number of good components manufactured in this batch : 22 .....

 --------Uploaded information on part production of batch no. 1 ---------

      <<<< SAWING cell >>>>
 Number of processed components   : 48
 Number of defective components   : 5
 Defect rate of the cell          : 9.52  %
 Efficiency of the cell           : 99.14 %

      <<<< MILLING cell >>>>
 Number of processed components   : 43
 Number of defective components   : 1
 Defect rate of the cell          : 2.56  %
 Efficiency of the cell           : 98.36 %

      <<<< TURNING cell >>>>
 Number of processed components   : 42
 Number of defective components   : 3
 Defect rate of the cell          : 7     %
 Efficiency of the cell           : 94.89 %

 ..... Number of raw components required for this batch      : 48 .....
 ..... Number of good components manufactured in this batch : 39 .....
```

Figure (6-5) Historical information on FMS cells.

CHAPTER SEVEN
Discussion and Conclusions

7.1 Introduction

Advances in technology are making the manufacturing environment increasingly complex. One of the fairly underestimated elements in the development of automated manufacturing is the specification, design, development and debugging of the overall control software. This is because the operational control of automated manufacturing systems is very complicated and involves accessing large static and dynamic data and complex algorithms. Currently, much of the overall control software for automated manufacturing systems is developed for each application. This imposes both high cost and dependency of the automated system to the specific organisation or vendors. At present, one of the most popular strategies for the overall control in many of such systems is the hierarchical control structure, where an upper level issues commands to a lower level and gets feedback on the achievement of these commands. However, this method of control encounters some disadvantages. This research, therefore, has identified the strengths and weaknesses of this type and other types (alternative control systems) of control systems for automated manufacturing systems. As a result, a hybrid control system was proposed and implemented on an experimental automated manufacturing system. This experimental testbed is the Flexible Manufacturing System at Durham university.

7.2 An Overall Control System for Automated Manufacturing

There are two types of dissimilar control systems which may be used for the overall control of an automated manufacturing system. These are hierarchical and heterarchical control systems. In the production system which is controlled by a hierarchical control system, the control modules are organised into levels of hierarchy. Here, the control system has a tree structure and different levels of control operate to different time scales and determine different operational decisions from aggregated parameters to more detailed ones. Two of the most popular hierarchical control models are the Automated Manufacturing Research Facility (AMRF) of the National Bureau of Standards (NBS) and the Advanced Factory Management and Control System (AFMCS) of the Computer Aided Manufacturing International (CAM-I).

The characteristics of a hierarchical control system include the following: In order to reduce the complexity and responsibility of the control functions in the system, levels are introduced. Therefore, the control problems are divided into manageable modules regardless of the complexity of the complete control structure. Descending from the top of the hierarchy, each level has a distinct planning horizon. In such a control system the sophisticated planning and scheduling is carried out at higher levels and the lower levels are left to execute the instructions received from the level above. For example, in the AMRF model the process planning and equipment programming are conveyed off-line. The shop controller is an order entry system. Scheduling is done at the cell level only. The workstation controllers receive commands from the cell level before executing them one at a time. Hence, there exists a strong master-slave relationship between the vertically adjacent entities, which may count as a disadvantage to such control systems. In addition, entities at one level are not allowed to communicate and exchange information with one another. As a result, each entity, in order to obtain information from its peer entity, has to invoke the control function at the level immediately above on the type of information it requires from the peer entity. This would have two disadvantages; firstly, too much information is passed between the vertically adjacent levels for the peer entities information exchange. Secondly, each level may spend a lot of time only on providing a means of information exchange for the peer entities while it could have utilised this time and processing capability on other control and decision making logics.

Since a crash at some level in a hierarchical control system will cause a serious disturbance to the entities below and may also bring the entire system to a standstill, researchers have tried to develop an alternative control system to overcome such weaknesses. This alternative control system is called a heterarchical control system, in which the intelligent entities negotiate with one another to plan, schedule and manufacture parts. Here, there is no direct control and no supervisor. All entities are equal and control decisions are reached through mutual agreement amongst the intelligent entities. Therefore, peer entities can exchange information and exercise direct communication.

However, since all of the control decisions must be reached as a result of negotiation and cooperation of the participant entities for part production, this tends to slow the system down in arriving at control decisions. As there is no supervisor, entities, in obtaining an overall view of the system, have to perform masses of information exchange and processing. This results in a high volume of traffic on the network to which all of the intelligent entities are interfaced. Hence, there existed a need for a control system which would take advantage of all of the good features of the aforementioned control systems.

7.3 The Proposed Hybrid Control System

A hybrid control system was proposed for the control of automated manufacturing systems. Such a control system utilises the best points of hierarchical and heterarchical control systems. Control is distributed by the movement of the centre of decision making from one controller to another. This is made possible by the capability for information transfer between controllers. In order to demonstrate the functionality and the practical real-time operation aspects of this hybrid control system, the Durham FMS was chosen as the system on which the proposed control model was to be implemented. The control system divided the entire FMS into three logical levels. At the top, the master level took charge of order entry, overall control and the scheduling of the system activities. Also, the user defined various system parameters and jobs at this level. The next level below was the cell level which was responsible for the scheduling and control of in-cell operations. Finally, there existed equipment at the lowest level, processing commands received from the levels above for successful part production.

The hierarchical nature of the system was demonstrated by the ability of an entity at the master level (i.e. the Master) to receive orders from the user before issuing control commands to the cell controllers residing at the cell level. The cell controllers then issued the subsequent control commands to the device controllers at the equipment level. In this way, is was shown that each level received some control command from its adjacent upper level. However, as it is believed that entities at the cell level should not be totally dependent upon the control commands being received from the master level, heterarchical behaviour was built into the system at the cell level. Each cell controller then only needed to receive the name of a job together with the batch size. How the parts were to be processed was the responsibility of the cell controllers, this exhibited the autonomal behaviour of the cell controllers.

An information transfer process was allowed among the cell controllers so that certain parameters which had specific and vital value in the cell controller decision making process could be obtained from the relevant entity directly. This feature made the overall system control exercise the decision making in a distributed manner (i.e. heterarchical control). For a cell controller to obtain information on the overall system performance so that it could formulate decisions for the benefit of the system as whole, two methods were possible. In the first method each cell controller had to exchange information with all other peer entities before drawing its own conclusions on how to achieve its goal within its cell. Utilising this method would have had two disadvantages. Firstly, there would have been an enormous amount of data flowing in the network (LAN) as a result of this and secondly, the control algorithms in each cell controller would have been slowed down in their decision making.

In the second method an entity at the master level (i.e. the Master) would have the most relevant information which could provide the basis for the cells' decision making. This method was applied to the Durham FMS where each cell controller received such information from the Master directly. For example, if a defective part was produced in a cell, all of the other cells enquired of the entity at the master level to acknowledge such an activity so that they could prepare themselves to produce one extra part than the original batch size to satisfy the requirements.

The hybrid control structure also has the potential to be utilised for the part processing of cells in the "bidding" mode. Although this method of part processing was not shown on the experimental system, there are indications such as, manufacturing cells are capable of exchanging information directly with one another, to make the control system suitable for such a method. The direct information exchange of the cells took place with the aid of a communication module namely, "`CellTalk`".

The technique used in `CellTalk` was demonstrated to be a network independent communication technique. It utilised a Common Address Memory in distributed Processors (`CAMP`). A series of communication routines was built within the `CellTalk` which allowed the direct memory access of the `CAMP`s in the entities. `CellTalk` highlighted the heterarchical behaviour of the overall control system. It was also found to be an effective tool for the debugging of the system software since it had a built-in monitoring system. This monitoring system allowed the user to get a snapshot of the dynamic database which held the cell and system specific data.

In order to apply the hybrid control model to the `FMS`, a System Integration Tool (`SIT`) was designed and developed. The `SIT` allowed the user to define its own manufacturing environment. It exhibited that the integration of equipment from a wide variety of vendors had become possible.

7.4 Integration of Multivendor Equipment

The development of the System Integration Tool resulted in the integration of programmable devices within the Durham `FMS`. The make, model and local programming language of each programmable device was different than the others. The `SIT` allowed the user to define and build the specific interface for each of the device controllers so that the communication, control and application of the hybrid control model could be exercised. The `SIT` was designed in such a way as to have several modules, with each module responsible for a certain aspect of the overall control system. This modular nature of the SIT made the software to be in a layered form where the interface of modules to each other is easily achieved.

Addition and elimination of manufacturing cells in the jobs were easily achieved by the user first entering the manufacturing database module where all of the system, jobs and related information were kept. The database provided the basis for the integration of multivendor equipment. The dynamic part of the database utilised an area of memory in the distributed processor. This CAMP based dynamic database allowed the system and cell specific data to be held in the memory locations and not on the disk, and provided an effective way of keeping and accessing the system and cell states. This method showed to have advantages over the common-memory based database system, as the writing of the cell states in the CAMP did not require the utilisation of the LAN. Therefore, incorporating this type of dynamic database reduced the data traffic on the network by half when it was compared with a common-memory based database. This may prove a valuable feature to the baseband LAN where the network has a small bandwidth.

The CAMP based database together with the CellTalk have the potential to be used in the integration of non-standard intelligent devices. For example, a special purpose-built manipulator for the destacking of a stack of flexible pre-cut material may require the cooperation and information exchange of the associated programmable table where the material is placed. The idea of a CAMP based dynamic database together with CellTalk could be employed for the integration of the aforementioned devices and also to the other devices in the system, if any exist.

7.5 Conclusions and Suggestions

The proposed control methodology was implemented on the experimental FMS. Tests showed that the proposed control theory functioned satisfactorily in real-time under laboratory conditions and produced a working system. The designed and developed System Integration Tool proved to be an effective tool for expressing the control method. The communication technique developed for the Durham FMS is a technique which is network and operating system independent. This technique, currently in the form of CellTalk, and the associated CAMP utilise a features which is specific to the LAN (Econet) in the Durham FMS. This feature being the easy access of the memories of the remote entities. However, this feature may not be present in other LANs. Therefore, the alternative to this would be utilising the Input/Output capabilities of the entities at the cell (cell controllers) and master levels.

Knowing that almost all intelligent devices have some sort of input output capability, a small size RAM (e.g. 1K) could then be plugged into the entities I/O boards (named as I/O RAM). By assigning a name (variable) to each memory location within the RAM, reading from or writing to the variables could have the same effect as that of CAMP and CellTalk. Procedures built into each entity would then be responsible in the read or write operation of the RAM via the I/O channels. Remote Procedure Call (RPC) would be used to gain access to the information on a distant entity.

The software modules developed for this in this research may be developed further to allow the following; The scheduling module to optimise the productivity of the machine tools, for example, by changing the sequence of operations throughout the cells. The system to cope with the production of several batches in a mixed fashion (batch mixing). A general purpose cell control builder which would intake the control definitions by means of some sort of user graphical interface, to build icons which represent specific control functions. The ideas in a LAN which is UNIX based (multi-tasking environment) to be applied and the software to be coded in a portable software language such as "C" or similar.

Currently, attempts are being made on the proposal of a research project based on further improvement of the proposed ideas here, which have attracted the attention of some of the small to medium size companies and research organisations. This initiative may result in obtaining a research grant (e.g. SERC/ACME) towards pursuing ideas which may benefit automated manufacturing systems.

REFERENCES

1. **Achatz, R. and Parrish, D.J.**, 1987, "Host Computer Controls FMS at all Levels", The FMS magazine, January, pp. 21-25.
2. **Acorn Econet**, 1983, Econet Advanced User Guide, Acorn Computers Limited, Cambridge.
3. **Adams, W.M.**, 1990, "Why 802.3 Ethernet LAN flourish in manufacturing operations", Computer-Integrated Manufacturing Systems, Vol. 3, No. 1, pp. 53-56.
4. **Albus, J.S., Barbera, A.J. and Nagel, R.N.**, 1981, "Theory and Practice of Hierarchical Control", Proceedings of 23 IEEE computer society international conference, Washington DC., pp. 18-39.
5. **Allan, B.**, 1989, "What Managers Really Need to Know About LANs" IEEE Network Magazine, November, pp. 15-19.
6. **Al-Qattan, I.**, 1990, "Designing Flexible Manufacturing Cells using a Branch and Bound Method", International Journal of Production Research, Vol. 28, No. 2, pp. 325-336.
7. **Alting, L., Christensen, S.C. and Pedersen, M.A.**, 1989, "An Integrated Miniature Laboratory for Research and Education", Software for Manufacturing, Proceedings of the Seventh International IFIP/IFAC Conference on Software for Computer Integrated Manufacturing, PROLAMAT'88, Dresden, 14-17 June 1988, Kochan, D. and Olling, G., Ed.:, Elsevier Science Publishers B.V., North-Holland, pp. 135-146.
8. **Anstiss, P.**, 1988, "The Implementation and Control of Advanced Manufacturing Systems", Control and Programming in Advanced Manufac-

turing, Rathmill, K., ed.:, IFS Publications Ltd., UK., pp. 373-386.

9. Avonts, L.H., and Van Wassenhove, L.N., 1988, "The part mix and routing mix problem in FMS: a coupling between an LP model and a closed queueing network", International Journal of Production Research, Vol 26, No. 12, pp. 1891-1902.

10. Bakker, H., 1988, "DFMS: A New Control Structure for FMS", Computers in Industry, Vol. 10, No. 1, March, pp. 1-9.

11. Beauchamp, K.G., 1987, "Computer Communications", Van Nostrand Reinhold, UK.

12. Bel, G., Bensana, E., Dubois, D., Erschler, J., and Esquirol, P., 1989, "A knowledge-based approach to industrial job shop scheduling", Kusiak, A., ed.:, Taylor & Francis.

13. Berra, P.B., Chen, C.Y.R., Ghafoor, A., Lin, C.C., Little, T.D.C. and Shin, D., 1990, "Architecture for Distributed Multimedia Database Systems", Computer Communications, Vol. 13, No. 4, May, pp. 217-231.

14. Bertok, P., Csurgai, G., and Haidegger, G., 1989, "The Integration of Intelligent Cell Controllers into Factory Networks", Proceedings of the Seventh International IFIP/IFAC Conference on Software for Computer Integrated Manufacturing, PROLAMA'88 Dresden, GDR., 14-17 June, 1988, ed. Kochan, D and Olling, G., pp. 445-452.

15. BITS 704, 1989, "ISO/IEC DIS 9506-1, Final text", MMS part 1, Service Definition, British Standard Institute.

16. BITS 705, 1989, "ISO/IEC DIS 9506-2, Final text", MMS part 2, Proto-

col Specification, British Standard Institute.

17. **BITS 706**, 1989, British Standard Institute "ISO/IEC Pre-DP 9506-5", MMS part 5, Programmable Controller Companion, British Standard Institute

18. **BITS 695**, 1989, "ISO/DP 9506-3", Robot Companion, British Standard Institute.

19. **BITS 696**, 1989, "ISO/DP 9506-4", NC Companion, British Standard Institute.

20. **Black, J.T.**, 1988, "The Design of Manufacturing Cells (Step One To Integrated Manufacturing Systems)", Proceedings of Manufacturing International'88, April 17-20, Atlanta, Georgia, pp. 143-157.

21. **Breeze, P.**, 1990, "The future for LANs", Automation, February, pp. 32-33.

22. **Brown, C.C. and Leonard, A.**, 1988, "Hierarchical Control Structure in Flexible Manufacturing Systems", Proceedings of the twenty-seven international Matador conference, April 20-21, Manchester, pp. 61-66.

23. **Browne, J., Harhen, J., and Shivnan, J.**, 1988, "Production Management Systems", Addison-Wesley.

24. **Bunce, P.G.**, 1988, "Developments in Advanced Factory Control and Management Systems", Control and Programming in Advanced Manufacturing, Rathmill, K., ed.:, IFS Publications Ltd., UK., pp. 425-444.

25. **Cheong, V.E. and Hirschheim, R.A.**, 1983, "Local Area Networks, Issues, Products, and Developments", John Wiley & Sons, England.

26. Coleman, J.R., 1988, "Will MAP Fly in US Factories?", Assembly Engineering, Vol. 31, Part 10, pp. 40-42.

27. Coll, J. and Allen, D., 1982, "The BBC Microcomputer User Guide", British Broadcasting Corporation.

28. Communiqe, 1990, "Aerospatiale Seeks MAP over Ethernet as a practical solution", Communiqe the Manufacturing Communication Journal, ComCentre, February, pp. 12-13.

29. Copas, C., and Browne J., 1990, "A rules-based scheduling system for flow type assembly", International Journal of Production Research, Vol. 28, No. 5, pp. 981-1005.

30. Daigle, J.N., Seidmann, A. and Pimentel, J.R., 1988, "Communications for Manufacturing: An Overview", IEEE Network, Vol. 2, No. 3, pp. 6-13.

31. Dickens, A. and Holmes, M., 1987, "The New Advanced User Guide", Adder Publishing Limited, Cambridge.

32. Drolet, J.R., Moodie, C.L., 1989, "A State Table Innovation for Cell Controllers", Computers and Industrial Engineering, Vol. 16, No. 2, pp. 235-243.

33. Duffie, N.A., Piper, R.S, Humphrey, B.J. and Hartwick, J.P., 1986, "Hierarchical and Non-Hierarchical Manufacturing Cell Control With Dynamic Part-Oriented Scheduling", Proceedings of fourteenth North American Manufacturing Research conference, May, Minneapolis, pp. 504-507.

34. Duffie, N.A. and Piper, R.S., 1987, "Non-Hierarchical Control of a Flexible Manufacturing Cell", Robotics & Computer-Integrated Man-

ufacturing, Vol. 3, No. 2, pp. 175-179.

35. Dwyer, J. and Ioannou, A., 1987, "MAP and TOP Advanced Manufacturing Communications", Kogan Page Ltd., UK.

36. Esprit, 1989, "The Project Synopses, Computer Integrated Manufacturing", Directorate General XIII, Commission of the European Communities.

37. Folts, H., 1988, "Open Systems Standards" IEEE Network, Vol. 2, No. 3, pp. 98-99.

38. Foote, B.L., and Ravindran, A., 1988, "Production Planning & Scheduling", Computers & Industrial Engineering, Vol. 15, Nos. 1-4, pp. 129-138.

39. Freer, J., 1988, "Computer communications and networks", Pitman computer systems series.

40. Frenzel, U. and Schubert, I., 1987, "Local Networks in Practice, US and German experience" Online publication, UK.

41. Gillespie, D., 1988, "What users need to do to make MAP a success", MAP/TOP user group summary, Long Beach, California, Vol. 3, No. 1, pp. 25-29.

42. Golenko-Ginzburg, D. and Sinuany-Stern, Z., 1988, "A Multilevel Production Control Model for FMS with Disturbances", International conference on Computer Integrated Manufacturing, May 23-25, New York, pp. 81-84.

43. Gordon, J. and Mclean, I., 1986, "The BBC MASTER 128 for high-flyers", Prentice-Hall International, London.

44. Greenwood, N.R., 1988, "FMS Programming and Control", Control and Programming in Advanced Manufacturing, Rathmill, K., ed:, IFS Publications Ltd., UK., pp. 389-398.

45. Groover, M.P. and Zimmerers, E.W., 1984, "CAD/CAM: Computer-Aided Design and Manufacturing", Prentice-Hall International, London.

46. Gupta, Y.P., Gupta, M.C. and Bector, C.R., 1989, "A review of scheduling rules in flexible manufacturing systems", International Journal of Computer Integrated Manufacturing, Vol. 2, No. 6, pp. 356-377.

47. Helmut, A. and Overbeeke, J., 1988, "Cellular Manufacturing: A Good Technique For Implementing Just-In-Time and Total Quality Control", Industrial Engineering, November, pp. 36-41.

48. Hohner, G., 1989, "Local Area Network Links Manufacturing Cells in Modular Growth Design", Industrial Engineering, July, pp. 23-28.

49. Hollingum, J., 1989, "Tooling Highway proves it out for Yamazaki", The FMS magazine, July, pp. 122-126.

50. Hughes, J.G., 1988, "Database Technology A software engineering approach", Prentice-Hall International, London.

51. Hutchison, D., 1988, "Local Area Network Architectures" Addison-Wesley England.

52. Hutchinson, G.K. and Chaturvedi, A., 1987, "Information Organization in Flexible Manufacturing Systems", Computer in Industry, Vol. 9, No. 4, pp. 353-368.

53. Hyer, N.L. and Wemmerlov, U., 1989, "Group Technology in the US

Manufacturing industry: A Survey of Current Practices", International Journal of Production Research, Vol. 27, No. 8, pp. 1287-1304.

54. Ioannou, A. and Dwyer, J., 1987, "MAP and TOP Advanced Manufacturing Communications", Kogan Page Ltd., London.

55. Jablonski, S., Ruf, T. and Wedekind, H., 1988, "Implementation of A Distributed Data Management System for Manufacturing Applications", International conference on Computer Integràted Manufacturing, May 23-25, New York, pp. 19-28

56. Jackson, S. and Browne, J., 1989, "An Interactive Scheduler for Production Activity Control", Vol. 2, Part 1, January, pp. 2-14.

57. Jones, A. and Saleh, A., 1990, "A Muliti-level/ Multi-layer Architecture for Intelligent Shop Floor Control", International Journal of Computer Integrated Manufacturing, Vol. 3, No. 1, pp. 60-70.

58. Ketchem, M.G., Smith, J.M. and Nnaji, B.O., 1988, "An Integrated Data Model for CIM Planning and Control", International conference on Computer Integrated Manufacturing, May 23-25, New York, pp. 338-342.

59. Kovacs, G.L., Bertok, P., Haidegger, G. and Csurgai, G., 1988, "Some Aspects of Reconfigurable Manufacturing Cells as Building Blocks of FMS", Proceedings of the second IIASA annual workshop on computer integrated manufacturing: future trends and impacts, July 18-20, Stuttgard, F.R.G., pp. 187-195.

60. Kusiak, A. and Heragu, S., 1988, "Computer Integrated Manufacturing: A Structural Perspective", IEEE Network, Vol. 2, No. 3, May, pp. 14-22. 62. Labs, W., 1989, "Cell Control: The Hardware, the

software, the controversies", I & CS, August, pp. 29-35.

61. Larin D.J., 1989, "Cell Control: What We Have, What We'll Need", Manufacturing Engineering, January, pp. 41-48.

62. Lyons, G.J., Duggan, J., and Bowden, R., 1990, "Project 477: Pilot implementation of a Production Activity Control (PAC) system in an electronics assembly environment", International Journal of Computer Integrated Manufacturing, Vol. 3, Nos. 3 and 4, pp. 196-205.

63. Maimon, O.Z., 1987, "Real-Time Operational Control of Flexible Manufacturing Systems", Journal of Manufacturing Systems, Vol. 6, No. 2, pp. 125-136.

64. Maley, J.G., Ruiz-Mier, S., and Solberg, J.J., 1988, "Dynamic control in automated manufacturing: a knowledge integrated approach", International Journal of Production Research, November, Vol. 26, No. 11, pp. 1739-1748.

65. Managaki, M., 1988, "Design Database Systems", NEC corp. Japan, Control and Programming in Advanced Manufacturing, Rathmill, K., ed.:, IFS publications, UK. pp. 13-35

66. Martin, J.M., 1989, "Cells Drive Manufacturing Strategy", Manufacturing Engineering, January, pp. 49-54.

67. Mcconnel, J., 1988, "Interworking computer systems", Prentice-Hall, USA.

68., McGregor, J. and Watt, A., 1983, "Advanced Programming Techniques for the BBC Micro", Addison-Wesley, London.

69. McGregor, J. and Watt, A., 1984, "The Art of Microcomputer Graphics for the BBC MICRO/ELECTRON", Addison-Wesley, London.

70. Mclean, C.R., 1988, "A Cell Control Architecture for Flexible Manufacturing", Technical Papers, Society of Manufacturing Engineers, Cell Controller Clinic, Schaumburg, Illinois, MS88-473, April, pp. 1-16.

71. MAP, 1987, "Manufacturing Automation Protocol", Version 3.0, July.

72. Miltenburg, J., and Sinnamon, G., 1989, "Scheduling mixed-model multi-level just-in-time production systems", International Journal of Production Research, Vol. 27, No. 9, pp. 1487-1509.

73. Nagarkar, S. and Bennet, D., 1988 "Flexible Manufacturing System Lets Small Manufacturer of Mainframes Compete with Giants", Industrial Engineering, November, pp. 42-46.

74. Nahavandi, S., 1990, "System Integration Tool at Durham University", Internal ACME/SERC report

75. Napier, R.G., 1984, "Networking With the BBC Microcomputer", Prentice-Hall International, London.

76. Nemes, L., 1987, "Intelligent Interfaces", Robotics and Computer-Integrated Manufacturing, Vol. 3, No. 2, pp. 171-174.

77. Nordsten, G., 1989, "FMS Aids Capital Turnover and Machine Usage", The FMS Magazine, January, pp.25-28.

78. O'Grady, P.J. and Lee, K.H., 1989, "An Intelligent Cell Control System for automated manufacturing", Knowledge-based Systems in Manufacturing, Kusiak, A., Ed.:, Taylor & Francis, UK., pp. 151-172.

79. PC Hardware and Software Guide, 1990, "Process Monitoring & Control", Control Engineering, Feburary, pp. 24-29.

80. Pitt, D., 1987, "Standards For The Token Ring" IEEE Network Mag-

azine, Vol. 1, No. 1, pp. 19-22.

81. Pleinevaux, P. and Decotignie, J.D., 1988, "Time Critical Communication Networks: Field Buses", IEEE Network, Vol. 2, No. 3, pp. 55-63.

82. Rabelo, L.C., and Alptekin, S., 1989, "Integrated Scheduling and Control Functions in Computer Integrated Manufacturing using Artificial Intelligence", Computers & Industrial Engineering, Vol. 17, Nos. 1-4, pp. 101-106.

83. Ranky, P.G., 1986, "Computer Integrated Manufacturing", Prentice-Hall International, London.

84. Ross, F.E., 1987, "Rings Are Round For Good!" IEEE network magazine, Vol. 1, No. 1, pp. 31-38.

85. Sarin, S.C., and Salgame, R.R., 1990, "Developent of a Knowledge-based system for dynamic scheduling", International Journal of Production Research, Vol. 28, No. 8, pp. 1499-1512.

86. Scheer, A., 1988, "CIM Computer Integrated Manufacturing", Springer-Verlag, Berlin.

87. Sepulveda, J.M., and Sullivan, W.J., 1988, "Knowledge Based System for Scheduling and Control of an Automated Manufacturing Cell", Computers & Industrial Engineering, Vol. 15, Nos. 1-4, pp. 59-66.

88. Shaw, M. and Wiegand, G., 1988, "Intelligent Information Processing in FMS", The FMS magazine, July, pp. 137-140.

89. Simpson, J.A., Hocken, R.J. and Albus, J.S., 1982, "The Automated Manufacturing Research Facility of the National Bureau of

Standards", Journal of Manufacturing systems, Vol. 1, No. 1, pp. 17-32.

90. **Street, Y.**, 1989, "Conversing Computers", Professional engineering, January, pp. 37-39.

91. **Sumpter, C.M., Gascoigne, J.D. and Weston, R.H.**, 1985, "Software Structures for Computer-Integrated Manufacture", Proceedings of the first National Conference on Production Research, University of Nottingham, September, pp. 86-94.

92. **Tangney, B., O'Mahony, D.**, 1988, "Local Area Networks and their Applications", Prentice-Hall International, London.

93. **Tchijov, I.**, 1989, "Flexible Manufacturing Systems (FMS): Current Diffusion and Main Advantages", Proceedings of the second IIASA annual workshop on computer integrated manufacturing: future trends and impacts, July 18-20, Stuttgard, F.R.G., pp. 223-247.

94. **TERCO CNC-1000**, 1980, "Theory book and Manual", TERCO AB, Sweden.

95. **Upchurch, K.M.**, 1988, "The use of relational data bases for real time data collection and process control", Robotics and Manufacturing, Proceedings of the Second International Symposium on Robotics and Manufacturing: Research, Education, and Applications, Albuquerque, New Mexico, USA, November 16-18, Jamshidi, M., Luh, J.Y.S., Seraji, H., Starr, G., Ed.:, ASME Press, New York.

96. **Vail, R.L.**, 1988, "Dynamic integration: some ideas from just-in-time manufacturing", Computer-Integrated Manufacturing Systems, August, Vol. 1, No. 3, pp. 179-185.

97. **Villa, A.**, 1988, "Distributed Architecture for Production Planning and Control in Discrete Manufacturing", International conference on Computer Integrated Manufacturing, May 23-25, New York, pp. 357-366.

98. **Wang, S.C.**, 1989, "Distributed FMS with Flexible Structure", Proceedings of the seventh international IFIP/IFAC conference on software for computer integrated manufacturing, PROLMAMAT'88, June 14-17, GDR, pp. 517-525.

99. **Watt, A. and McGregor, J.**, 1983, "The BBC Micro Book, BASIC, Sound and Graphics", Addison-Wesley.

100. **Weatherall, A.**, 1988, "Computer Integrated Manufacturing", Butterworth & Co Ltd.

101. **Wemmerlov, U. and Hyer, L.**, 1989, "Cellular Manufacturing in the U.S. Industry: A Survey of Users", International Journal of Production Research, Vol. 27, No. 9, pp. 1511-1530.

102. **Westerdale, G.D.**, 1988, "Implementing a Manufacturing Resources Planning (MRPII) Information System in a Job Shop Environment", Proceedings of Manufacturing International'88, Atlanta, Georgia, April 17-20, Vol. 3, ed. Chryssolouris, G., Turkovich, R.W., and Francis, P., pp. 279-287.

103. **Weston, R.H., Sumpter, C.M. and Gascoigne, J.D.**, 1986, "Industrial computer networks and the role of MAP, part 1", Microprocessors and Microsystems, Vol. 10, No. 7, pp. 363-370.

104. **Weston, R.H., Gascoigne, J.D. and Sumpter, C.M.**, 1987, "Industrial computer networks and the role of MAP, part 2", Microproces-

sors and Microsystems, Vol. 11, No. 1, pp. 21-34.

105. Weston, R.H., Hodgson, A., Gascoigne, J.D., Sumpter, C.M., Rui, A. and Coutts, I., 1989, "Configuration methods and tools for manufacturing system integration", International Journal of Computer Integrated Manufacturing, Vol. 2, No. 2, pp. 77-85.

106. Williams, C., 1988, "MMS to DIS - What's the impact?", MAP/TOP user group summary, Long Beach, California, Vol. 3, No. 1, January, pp. 151-160.

107. Vollmann, T.E., Berry, W.l., Whybark, D.C., 1988, "Manufacturing Planning and Control Systems", Dow Jones-Irwin, Illinois.

108. Wright, J., 1987, "Bringing Networks to the Factory", Engineering Computers, pp. 56-58.

109. Zimmerman, B., 1980, "OSI reference model - the OSI model of architecture for Open System Interconnection", IEEE Transaction on Communication, Vol. 28, pp. 425-432.

APPENDIX A
Design of Interface Module for Programmable Devices

A.1 Design of an Expansion Board for BBC Microcomputers

Since there was no commercially available expansion board for the BBC microcomputer which would provide the specific requirements demanded by the application of BBC as cell controller in a FMS environment, expansion boards were designed and implemented.

The BBC microcomputer has been designed to allow expansion in a number of ways. The USER port and the 1 MHz bus can be used to add extra hardware to BBC. With the USER port there exists a problem of limited input/output lines (only 8 lines and two control lines). Alternatively, the 1 MHz bus can be used for more complex peripherals, as it allows direct access to the CPU address and data bus.

An expansion board was designed to provide four 8-bit bidirectional lines with the capability of sending and receiving up to 16-bit digital signals in parallel. The board required a 1MHz bus connector, an address decoder, a clean up circuit as recommended by Acorn Computer Limited, a data buffer, two Versatile Interface Adaptors (VIA) 6522 and three octal bus transceiver and registers.

The connection to the BBC is done via a 600mm length of ribbon cable terminated with a 34 way IDC socket, and fitted with strain relief. The 1 MHz bus carries the following signals: R/NW (Read Not Write); 1 MHZE; NNMI; NIRQ; NRST; NPGFD (not used); NPGFC; ANALOG IN (not used); eight address lines, A0 to A7 and eight data lines, D0 to D7 and several ground return lines. A full address within the BBC is made of 16 bits, and a convenient way of considering is to split it into two halves. The upper half defines a page address with each page containing 256 bytes, and the lower half defining a particular byte within the page. With this representation, memory area FRED, which contains only addresses beginning with FC (hex), is the page FC.

Since the decoded signals are active low, the bus page select signals is coded as NPGFC, meaning 'not page FC'. The drive circuit on the bidirectional data bus are arranged to be active only when NPGFC is present, and then only the low eight address lines, A0 to A7, are required to define an address within the page. The R/NW signal is used to establish the direction in which the data bus drivers operate. A high will transfer data to the computer and a low, from it. Page FC (hex) is reserved for peripheral with small memory requirements and memory block from FCC0 (hex) to FCFE (hex) is allocated for general purpose use.

Figure (A-1) illustrates the designed expansion board. Since the Not Interrupt Request (NIRQ) signal on pin 8, and the Not Non-Maskable Interrupt (NNMI) signal on pin 6, are both active low and open collector, pull-up resistors of 3 $K\Omega$ are used.

Because there is more than one VIA on the expansion board, address decoder (SN74LS137) is used to address each VIA. The two chip select input signals on the VIA, $CS1$ and $\overline{CS2}$ are connected to the 1 MHz bus address lines via the decoder and the NPGFC2 via the clean up circuit (CNPGFC2) respectively. Address lines A7 to A5 are used as input to the decoder and by selecting FCA0 (hex) and FCE0 (hex), a low output on Y_5 and Y_7 pins will be produced respectively. These signals are used as input to the $\overline{CS2}$ pins of the VIAs (VIA 0 and VIA 1).

As recommended by Acorn, the CNPGFC signal, together with the lower 8 address lines, are decoded to select the VIAs. The BBC microcomputer's CPU normally operates with a 2MHz clock, but with a slow-down circuit which has the effect of stretching the "clock high" period immediately following the detection of a valid 1 MHz peripheral address Fig. (A-2). Two problems are manifested as a result of this.

Firstly, address will change and may momentarily become 1 MHz address while the 2 MHz CPU clock is low, but while the 1 MHzE signal is high. This could give rise to address decoding glitches labelled 'P' and 'Q'. The glitches are not normally important because the 1 MHzE clock is then low. The 'P' glitches can cause problems because the 1 MHzE signal is then high. Spurious pulses may therefore occur on the various chip select pins.

Secondly, if the CPU deliberately addresses a 1 MHz peripheral during the period that 1 MHzE is high, the device will be addressed immediately, and then again when 1 MHzE is next high; this is because the CPU Clock will be held "high" by the stretching circuit until the next coinciding falling edge of the 1 MHz and 2 MHz clocks. These are marked as 'R' and 'V' in, Fig. (A-2). In many cases this double accessing is not a problem except when reading from or writing to a location twice has some additional effect. For example, clearing an interrupt flag prematurely. This effect means that the 1 MHzE bus cannot be used as a conventional "address valid" signal. However, addresses will always be valid on the rising edge of 1 MHzE.

In order to overcome the problems mentioned above, a 'Clean' version of NPGFC signal has to generated (i.e. CNPGFC). Figure (A-3) shows a "clean up" circuit recommended by Acorn to cater for this, and describes the functions as follows; before CNPGFC can go low, a valid page address with 1 MHzE low must occur. The page low is then latched into a D-type flip-flop on the rising edge of 1

MHzE clock. The CNPGFC signal will go low a time flag (40 nS) after 1 MHzE goes high and it will remain valid until 40 nS after 1 MHzE has gone low again.

An inverter (SN54LS04) is used to invert the CNPGFC signal which is used as a high input for the $CS1$ pins of the VIAs. Since all the logic lines will have to be buffered for each peripheral an octal bus transceiver with tri-state outputs (SN74LS245) is used as a data buffer, and the pin for its data direction DIR (pin 1) uses the inverted R/NW signal.

The VIAs which are interfaced are SY6522 Versatile Interface Adapters. They are very flexible input/output control devices. They each contain a pair of 8-bit bidirectional I/O ports, two 16-bit programmable timer/counters, serial data port, latched output and input registers, expanded "handshake" capability which allows positive control of data transfer between processor and peripheral devices with 1 MHz operation.

All the signals on port A and B of VIA 1 and port B of VIA 0 are buffered through a SN74LS245. The socket for these components can serve a dual purpose since they can accommodate an octal bus transceiver and register (SN54LS646) as well (only one IC can be fitted per socket at a time). The 646 IC is chosen to increase the expansion boards capability and possesses the following features; independent registers for A and B buses, multiplexer real-time and stored data, three-state output and true logic data path.

One of the advantages of utilising 646 is that data can be stored on the buses of two separate 646 IC and then all 16-bits of data from two IC can be sent out in parallel. In this way data is sent as 8-bit at a time from the BBC but stored on two separate 646 before being released as 16-bit in parallel. The data direction for both buffers (245 and 646) can be set in two ways; hardware or software. To set data direction in hardware the provided links on the board can be utilised. This is done by linking pin number 3 of the socket into which the buffer is inserted to 5V or 0V, Fig. (A-1). Setting the data direction in software requires pin number 3 to be linked to pin marked I/0. The pin number 21 ($\overline{G}$) also can be set either in software or in hardware to control the 646 functions.

Port A on VIA 0 is used for the software control of 646. Software can be developed to send a high or a low to control the functionality of 646 buffers. At preset only the 245 buffers are implemented on the expansion board. Note that when a 245 buffer is inserted into the provided socket, pin 1 on the IC must be positioned against pin 3 of the socket and pin 22 should be linked to 5V.

A $0.1\mu F$ capacitor is used as a decoupling capacitor across the 5V and 0V of all ICs on the expansion board. In addition, a socket is provided on each of the VIAs ports to external devices for the insertion of pull up or down resistors, as may be required in future.

A.2 Expansion Board Functionality Test

Once the expansion board was manufactured it had to be tested to ensure its correct functionality. In order to carry out the test the recommended 'clean up' circuit programs were developed to access the 1 MHz bus. Unfortunately, the clean up circuit did not work as a result of incorrect timing, Fig. (A-4.1). Comparing this with the given timing diagram for the 1 MHz bus by Acorn, Fig. (A-5) it soon becomes clear that when the page FC is selected, the state of the 1 MHzE clock should go from a high to a low and then a high. However, for some unknown reason, the 1 MHzE clock signal appeared to be inverted.

To make the board function correctly the OR gate from the clean up circuit was removed. As a result of this, the width of the cleaned page select signal was increased, Fig. (A-4.2), but the board still was not functioning. Data was read or written only when the 1 MHzE clock signal was low. The only way to achieve the desired result was to invert the 1 MHzE clock signal before inputting it into the clean up circuit, Fig. (A-4.3). This shifted the CPGFC signal by half a micro second to the left, coinciding with the low cycle of the clock. By now the board was functioning and there were no problems when the ports on VIAs were being used as output.

When an attempt was made to read the VIAs ports however, they would not function correctly at all times. It was discovered that the VIAs were accessed more than once on one read cycle and that this had been caused by the glitches on the page select signal.

At this point it was decided that a different clean up circuit should be designed rather than sorting out the problem with the recommended one. A first-order low pass filter was designed to cater for this need. After a series of tests the values for R and C were found by trial and error to be 1 $K\Omega$ and 330 pF respectively. The original clean up circuit was replaced by the RC filter, Fig. (A-6).

The low pass filter is comprised of a resistor and a capacitor arranged in series. The cutoff frequency of the filter f_c at $-3dB$ is calculated to be 0.48 MHz. All the frequencies below 0.48 MHz are passed and above that are attenuated. The product RC is known as the time constant of the circuit and that is the time taken for the output to reach 63% of input voltage. The rise time t_r, which is the time taken for the voltage to rise from 10% to 90% of its final value, is calculated to be $0.73\mu s$. The fall time is defined in a similar way to rise time; it is the time for the level to fall from 90% to 10% of its original level and as with rise time

$t_f = 2.2RC.$

A CMOS 2 input NAND Schmitt trigger is used as a means of converting the sine waves (V_{out} into a square wave with fast rise and fall times. The timing diagram for page select signal after the implementation of the low pass filter is shown in Figure (A-7).

A.3 Test Programs for the Expansion Board

A series of programs are developed to test the expansion board and the logic levels are monitored at various points on the board (program BOARD-TEST, Appendix F) . To test the ports on the VIAs configured as output, the program will first select the 1 MHz bus before initialising the VIA. A particular port is then selected and configured as output. Next, the program outputs signals to that port and a digital storage scope may be used to monitor the arrival of the signal. To conduct the test on the VIA ports, configured as input, an absolute encoder may be used. The encoder can be set to a fixed position and that position can be read continuously. The read value can be checked against a preset values (255) each time. Should there appear a glitch or spike in the page select signal (CPGFC), then the device is not accessed for a short period. This will cause a value of 255 to be read by the computer, and terminate the execution of the program. This test was carried out and after running the program for six hours continuously, it became evident that the RC network is fuctioning correctly.

A.4 Design of a Remote Control Board for the CNC Miller

For a programmable device to be able to contribute the flexibility that is expected in an FMS environment, its capabilities should include; being able to receive the machine control commands from the cell controller and execute them successfully, the ability to communicate with its cell controller and provide it with the signals which are critical to machine operation (i.e. feedback signals), allow for the UPLOAD and DOWNLOAD of files to and from its cell controller.

In order to endow the CNC miller with some of these capabilities, its control circuit was slightly modified. This can be divided into two parts, one where the designed interface enabled the miller to receive the control command directly from the cell controller, and the other making the remote resetting, initialising and jog control of the mill's axes, possible.

The circuit diagram for the mill controller was carefully studied. The keyboard on the mill's control panel (function keys) are encoded through a keyboard encoder interface (MM5740AAE) before being sent to the PIA (6820). An interface was built to bypass the keyboard encoder allowing the cell controller to send the encoded data directly into the PIA's data bus, Fig. (A-8). A manual (toggle) switch was installed at the mill's control panel to disable or enable the keyboard, allowing remote or manual operation of it.

Table (A-1) shows the way in which the keyboard is encoded. This information is worked out from the MM570AAE data sheet together with the TERCO-1000 keyboard circuit diagram. Once the keyboard is disabled, codes can be sent directly to the PIA, for example, sending 8B (hex) will have the same effect as that of the START key being depressed on the mill's controller.

Procedures are built within the cell controller software which take care of sending the right code to the mill's controller for various control commands. Data Codes sent from the cell controller is routed via the expansion board to mill's PIA. Prior to commencing a milling operation, its datum must be defined. This is carried out by sending the table to a prefixed (home) position. Micro switches are installed on the mill's table axes. Feedback signals from these switches indicate whether or not the table is in its home position.

Table (A-1) Encoded data for CNC-1000 function keys

Function key	*Matrix address*		*Encoded value* (Hex)	*Function key*	*Matrix address*		*Encoded value* (Hex)
	x-value	y-value			x-value	y-value	
1	9	10	B1	S	8	4	53
2	8	10	B2	T	6	3	D4
3	7	10	33	X	8	6	D8
4	6	10	B4	Y	5	3	59
5	6	1	35	Z	9	6	5A
6	5	1	36	Block by Block	2	2	8D
7	5	10	B7	Change	2	8	09
8	4	10	B8	Dump Tape	9	7	82
9	4	1	39	End of Prog	3	10	3A
0	3	1	30	End of Block	3	2	0A
-	2	10	2D	Erase Block	9	9	84
+	3	5	BB	Exam	2	9	88
F	6	4	C6	Insert Block	8	1	05
G	6	5	47	Load Tape	9	5	81
I	4	2	C9	Prog	2	6	8E
J	5	4	CA	Read	8	5	87
K	4	4	4B	Set Zero	8	3	06
M	5	6	4D	Start	2	5	8B
N	5	7	4E	To Zero Point	9	8	03

In order to send the table to its home position, an interface is built into the Jog circuit of the mill's controller. Signals are sent directly from the cell controller to move the table to the desired position (home). Two Micro switches (Single-pole Double-throw) are used on each axis, one as a limit switch providing "input" signals to the cell controller and the other as a means of cutting off the input signals (circuit breaker) to the stepper motor of the axis on which the limit is reached. For higher reliability a Double-pole Double-throw switch can be used instead of the two switches, but this is more costly.

Since the ports on the cell controllers VIA are normally at high impedance, a single micro switch could not have been used for both sensing the tables travel limits and as a circuit braker for the input signal to the stepper motors unless additional circuitory was designed. In order to simplify the problem two switches were used. Although it is not suggested that this is the best solution, it has been proved to work successfully.

The table can be sent to its home position either by the manual jog control switch or by the cell controller, sending signals directly to the jog control interface. In both cases, once the limit is reached on an axis the micro switch (circuit breaker) will automatically stop further movement of the stepper motor on that axis.

To send the table to its home position, the cell controller sends the appropriate signal to jog control interface and reads the state of the limit switch, i.e. whether the limit has been reached. Once the limit has been reached, the first micro switch (circuit breaker) performs a hardware stoppage of further signals to the stepper motor, while the second one prevents the cell controller software from sending further signals. Since at the end of a milling operation table will travel to its home position, the limit switch is used for the acknowledgement of the end of the milling operation.

An interface was also built into the reset line of the mill controller, so that the reset of the controller can be performed directly from the cell controller.

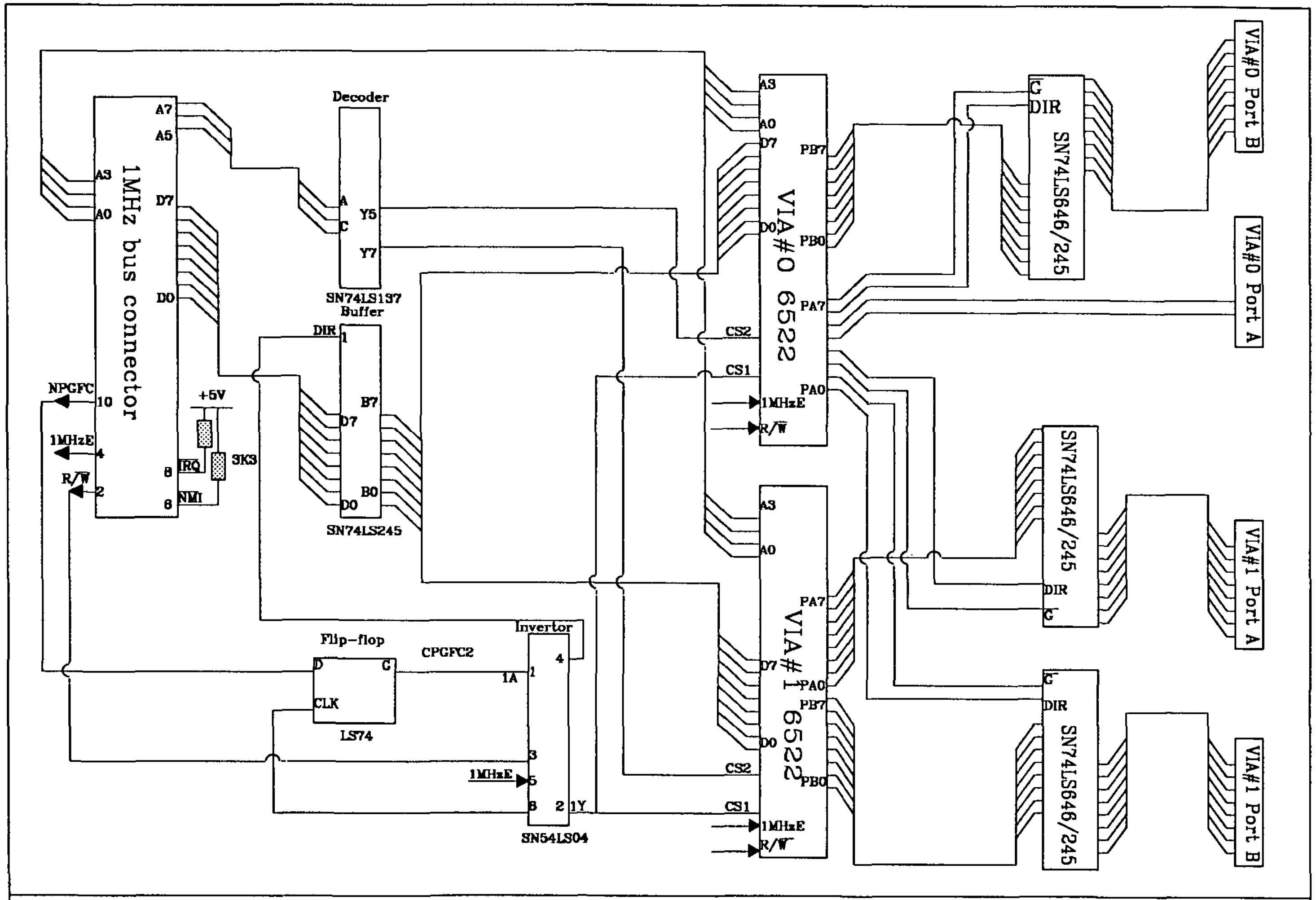


Figure (A-1) The cell controller interface module.

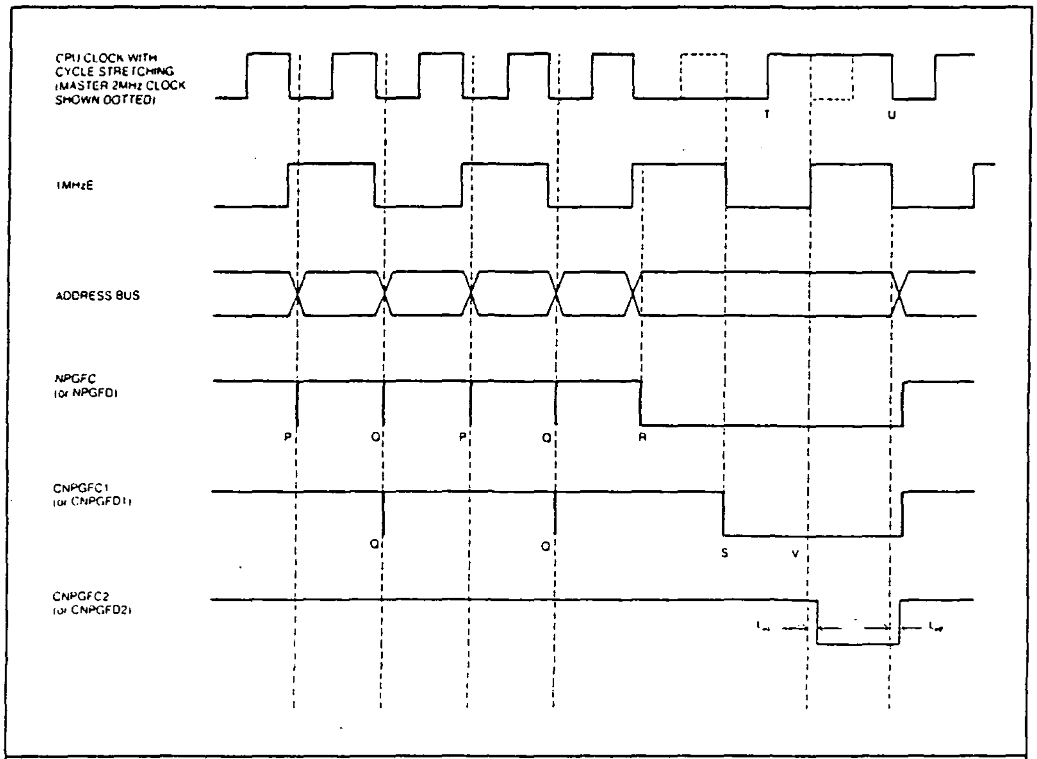


Figure (A-2) 1 MHz bus timing showing page select signals.

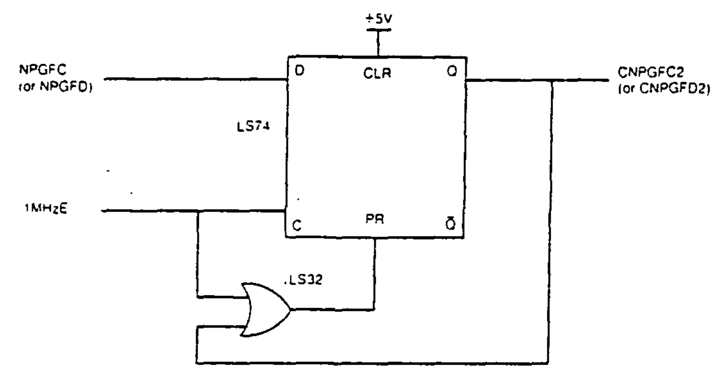


Figure (A-3) The Clean up circuit.

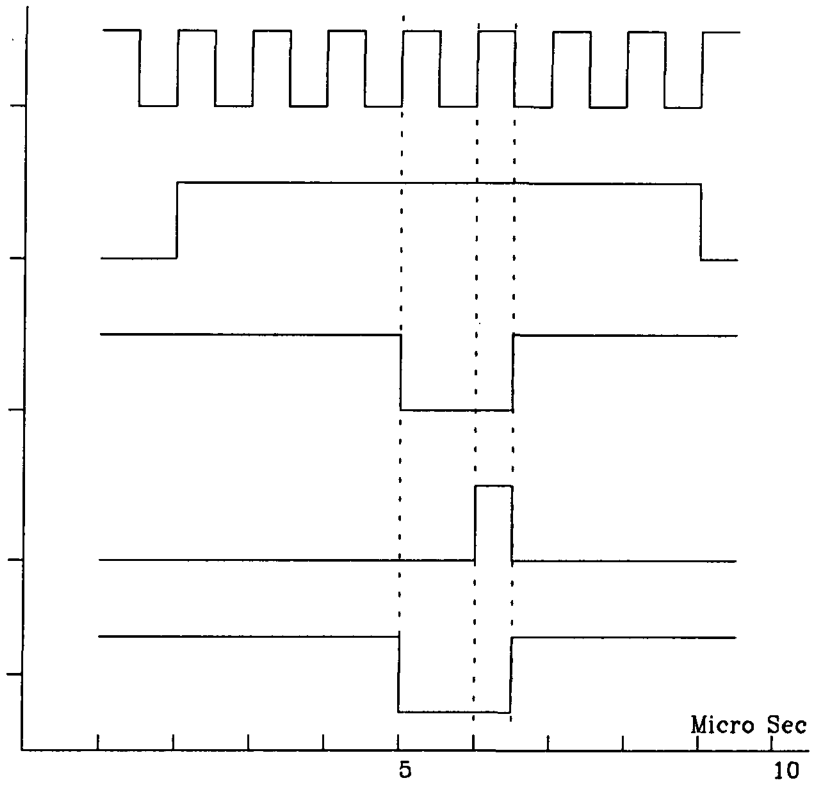


Fig. (A-4.1) Using the BBC recomended Clean-up circuit.

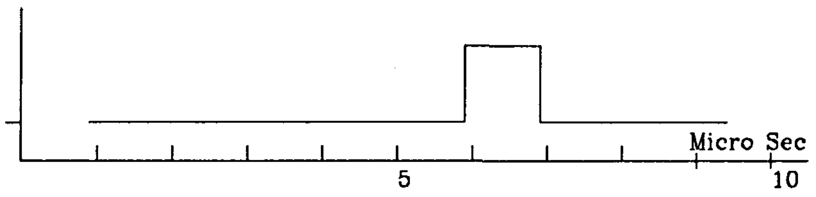


Fig. (A-4.2) Removing the OR gate.

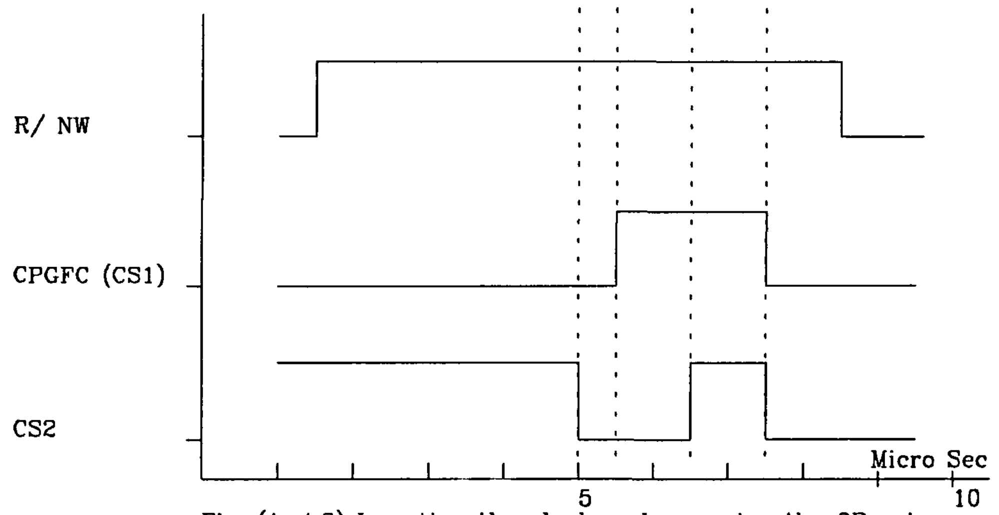
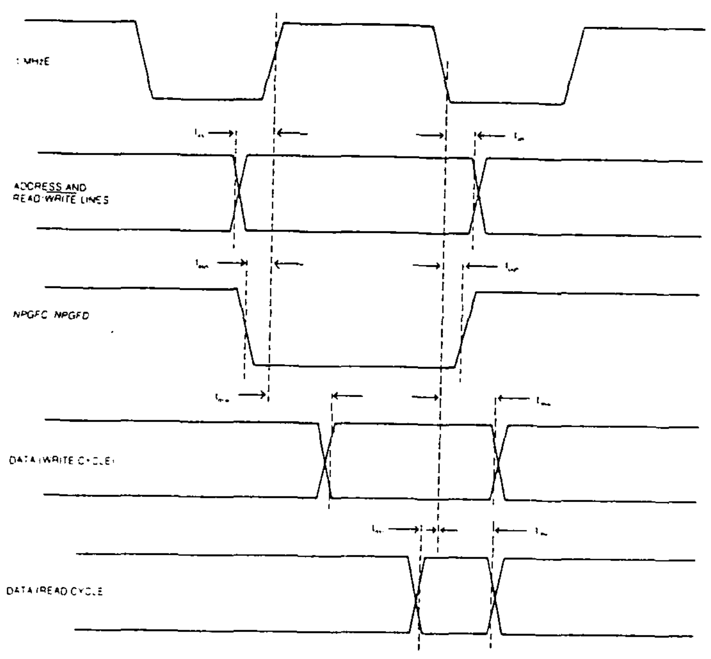


Fig. (A-4.3) Inverting the clock and removing the OR gate.

Figure (A-4) Derivation of valid page select signal.



The timing requirements are:

Description	Symbol	Min.	Max.
Address (and Read/Write) Set-up time	t as	300nS	1000nS
Address (and Read/Write) Hold time	t ah	30nS	-
NPGFC & NPGFD Set-up time	t pgs	250nS	1000nS
NPGFC & NPGFD Hold time	t pgh	30nS	-
Write data set-up time	t dsw	-	150nS
Write data hold time	t dhw	50nS	-
Read data set-up time	t dsr	200nS	-
Read data hold time	t dhr	30nS	-

Figure (A-5) The 1 MHz bus timing.

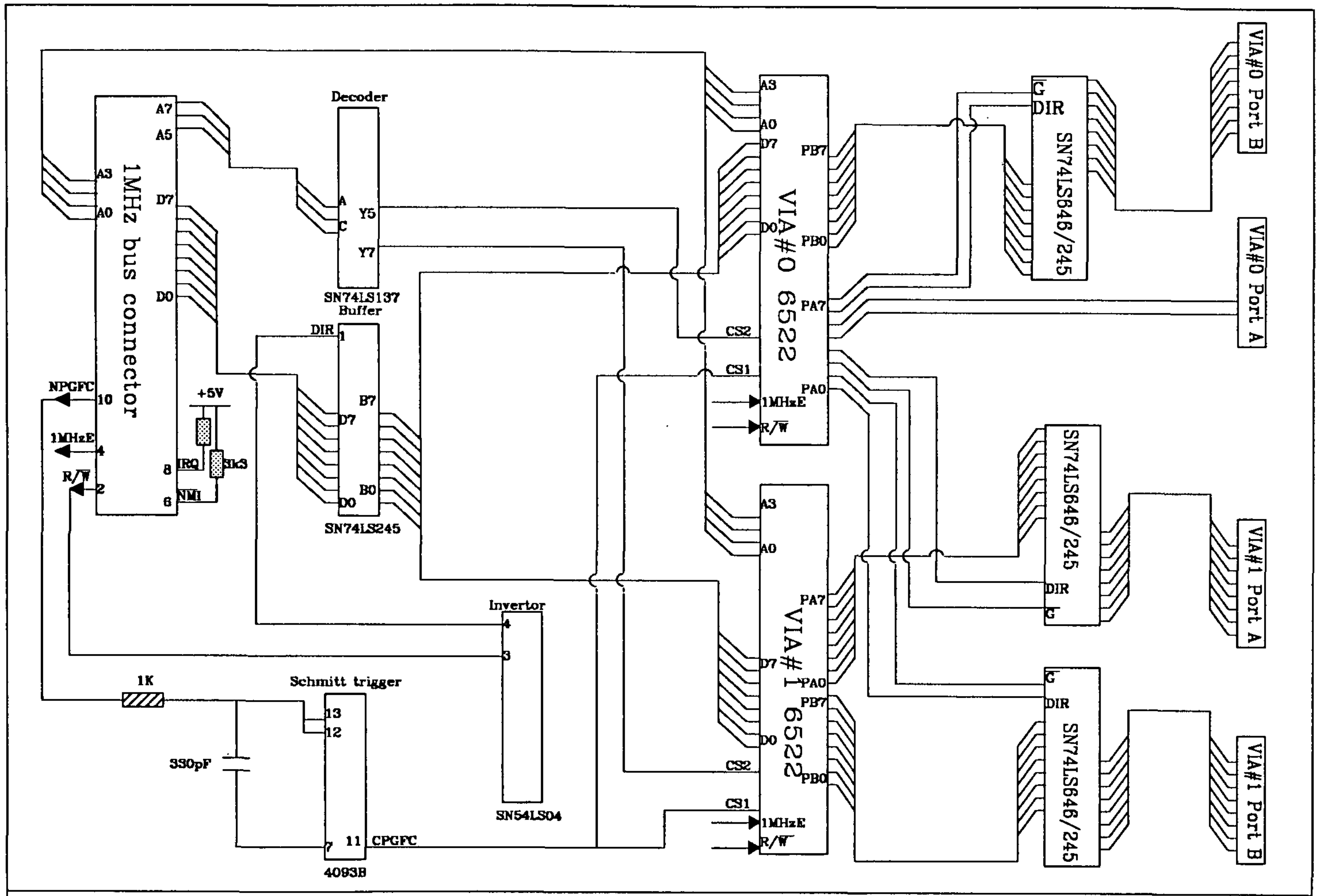


Figure (A–6) The modified circuit for the interfacing module.

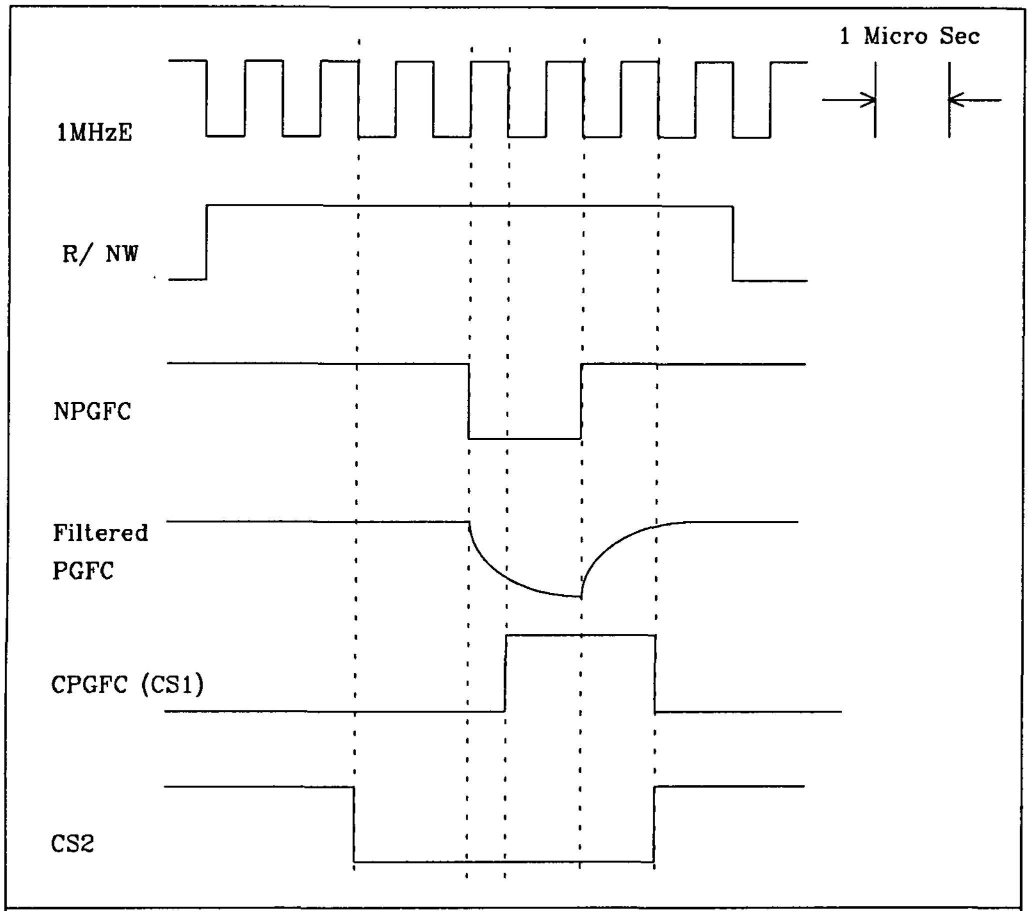


Figure (A-7) Derivation of valid page select signal using a first order filter.

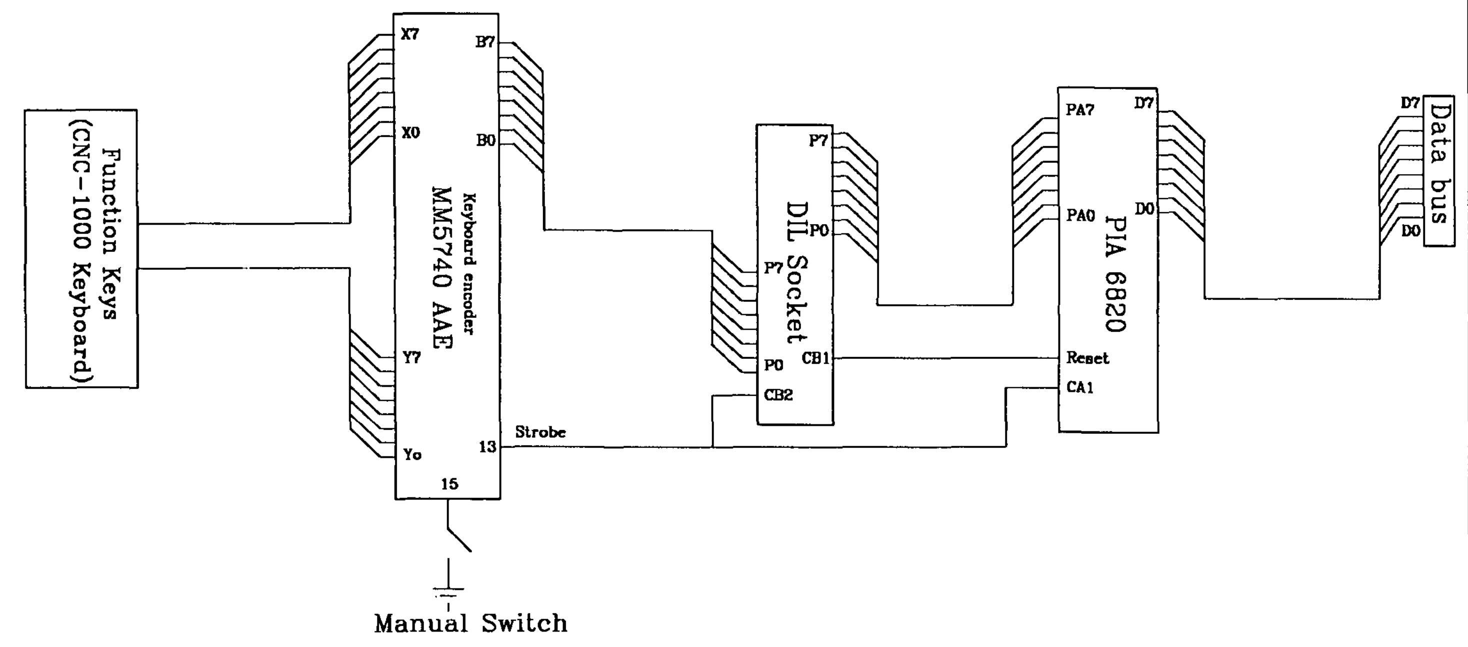


Figure (A-8) The keyboard encoder bypass circuit.

APPENDIX B

Data Organisation and Representation in FMS

B.1 Manufacturing Database

The functionality of the manufacturing database is very similar to some of the commercially available databases running under the DOS (e.g. SMART). It provides the user with a simple Structured Query Language (SQL). Some parts of the routines within the manufacturing database, Appendix F, utilise a modified version of codes listed in [Gordon 1986] and [McGregor 1983]. Although there is no limit to the number of records of a datafile and the number of fields of its records, to save disc space the maximum number of records and fields was set to five and eighteen respectively at the start of the program.

Records of a data file are related by a field (keyfield) and the contents of the keyfield from different records together with their record numbers are stored on a separate file namely 'index file'. To speed up the operation of the database a two dimensional array "index table" is used to hold the content of index file while the database is being used. The user is provided with a "$" prompt and the main menu will simplify utilisation of the database.

The program will only accept those command lines defined by the database and the order in which the commands within a command line are entered is also important. Each command line calls one, or a number of, procedures. At this stage it is assumed that the user has some knowledge of database and has a plan of how to define the structure of records of relation.

At the beginning of the program user is given the choice of creating a new job. If this option is chosen, the user is then required to enter the new job name.

The program will then create a data file for the new job together with an index file. The name of the index file is encoded by adding "IND" to the end of the job name. Procedure PROCcreate-new-job is called to perform this. Users may refer to the menu for the facilities provided by the database by entering letter "M" at the prompt. The structure of a manufacturing database shown in the form of a flowchart in Figure (B-1).

Records may be added to an existing datafile by entering 'ADD' command at the prompt. This will call the procedure PROCadd-rec where the user can then enter data to the fields of a record. Each record is then added to the end of the datafile and an entry is made to the index table. Once user has finished with a datafile, it can be closed by entering close command followed by the name of the datafile (CLOSE 'filename') which is performed by the procedure PROCclose-file. However, closing a data file does not update the content of its index file. To do this the close command has to be followed by the name of the datafile, update command, index filename and the field number of the keyfield (CLOSE 'filename' UPDATE 'index filename' 'field number'). ROCclose-update is used for the closing of a datafile and updating of its index file.

Records of a datafile may be edited after being created. To do this a record number has to be preceded by edit command (EDIT 'record number'). PROCedit-rec will display the fields of the record to be edited and user the is then given the choice of changing the content of each field or leaving it unchanged.

Records of a datafile can be marked for deletion by entering command followed by the start and the finishing record numbers separated by a semicolon (DELETE 'from record' ; 'to record'). PROCdelete performs this by simply marking the records to be deleted. The subsequent deletion is performed by entering the compress command followed by the name of the datafile (COMPRESS 'filename'). PROCcompress-recs will read the records of a datafile and overwrite the records which have been marked by the delete command. It also removes the entry of deleted records from the index file.

An option is given to the user to recover the records which have already being marked by the delete command. This is done by entering the retrieve command followed by the record number at the prompt (RETRIEVE 'record number'). A deleted record can only be recovered if the datafile has not been compressed. PROCretrieve-rec performs this by simply removing the deletion marks from the front of the record, and updating the index file.

In order to prevent the accidental modification of a datafile, user is provided with two options. One, where an existing datafile may be opened for reading purpose only, in which case the open command is followed by the data filename, index command and index filename (OPEN 'filename' INDEX 'index filename'). PROCopen-file is called to open a file without updating it. The other option allows user to open a datafile for updating. The datafile must be the one which is already in existence and the open command must be followed by update command, data filename, index command and the index filename (OPEN UPDATE 'filename' INDEX 'filename'). This is performed by PROCopen-update-file. The open command reads the structure of the datafile into memory and by putting the content of the index file into memory allows user to perform data management activities.

Once a datafile is opened the record pointer is set to one by default. However, it is possible to change the position of the record pointer to any existing record by entering the goto command followed by the desired record number (GOTO 'record number'). PROCgoto accepts both positive and negative numbers to move the record pointer forward or backwards.

Records of a datafile can be listed by entering the list command (LIST). This command lists all records of the datafile and the output to the screen is controlled by the user. PROClist-recs performs this and it waits for a key to be pressed by user on printing of every twentieth line to the screen. Displaying of a single record is also possible by entering the display command followed by the desired record number (DISPLAY 'record number'). If no record number is given, then the record to which the record pointer is pointing will be displayed. PROCdisplay-rec is used for the displaying of randomly selected records by user. Viewing a specific field of any record within a datafile can be performed by using the show command followed by the record number and the field number (SHOW 'record number' 'field number').

Organising records of a file in a particular order requires sorting. Records can be logically ordered so that a particular field (keyfield) appears in alphabetical order (assuming the field type is of characters). Alternatively, they may be ordered so that a numeric field appears in ascending or descending order. Index files can then be referred to as being sorted 'ON' a particular field (keyfield).

The order in which the records of a datafile are displayed is determined by the content of its index file. A datafile may have several index files where the record's relations are stored. Hence, the logical order of the records depends on which index file is in use. To create a new logical order for the records of a datafile, the order command can be entered, followed by 'ON' command, the field number which is to be chosen as the keyfield, 'TO' command and the name of the index file where the new index data is to be written (ORDER ON 'field number'

TO ‘index filename’). This is performed by PROCorder-recs. PROCorder-recs will create a new index file and, after organising the record in the order which is based on the keyfield, will write the content of the keyfield form different records, together with their record numbers, to the index file.

During an ordering session the data fields (keyfield) of the records are being sorted by PROCsort. This utilises a bubble sort [McGregor 1983] which sorts the fields stored in the index table into the required order. Considering an array of data with n entries to be sorted by bubble sort (in ascending order), the algorithm will decide each time whether the value of entry n is smaller than the entry n-1, if so the two entries are swapped round (PROCswap). This process is repeated until the data array is sorted.

Search for a record can be made among the records of a data file, being ordered on a keyfield, to see if there is an entry corresponding to the content of the field being searched for (searchspec). This is carried out by entering the find command followed by searchspec (FIND ‘field content’). PROCfind-rec utilises a searching technique known as "logarithmic search". This method of searching requires examination of approximately Log_2 n entries (fields) before finding the desired one. It begins examining the entry in approximately the middle of the index table (PROCchop-arraytable). If the given searchspec is the same as the examined entry, the search is terminated successfully. If the given searchspec comes before the examined entry, the first half of the index table must be searched next, otherwise the second half of the index table must be searched. This process is repeated until the searchspec is found.

There are some procedures which are being called from other procedures. The ones which have not yet been mentioned are; PROCread-index, PROCget-num and

PROCget-chars. PROCread-index is used by 'OPEN' command, and it reads the content of an index file into the index array. PROCget-num and PROCget-chars are used by the 'EDIT' and 'ADD' commands. When a datafile is created, the fields of its records can either accept numeric or alphanumeric (characters) and this information is held in the header record of datafiles. When new records are to be added or the existing one edited, depending on the fields type, PROCget-num or PROCget-chars is called.

B.2 From Design to Manufacture

A drawing package namely "CAD10" has been developed to allow for simple shapes to be designed and then this data is postprocessed for the manufacturing phase. The postprocessing of design data is carried out for the CNC-1000 miller. Figure (B-2) illustrates a flowchart of the program. The routines which deal with drawings utilise information on graphics in BBC microcomputers from [McGregor 1984], [Watt 1983]. At the beginning of the program, user is asked to enter the part (workpiece) information such as its length, width, first depth of cut and feedrate and what the generated part-program is to be called. This is performed by procedure PROCpart-info.

Next the screen is divided into two windows, one where the menu is present (text window) and the other where the drawing (graphic window) takes place. The milling machine utilised for the FMS can only accept parts (workpieces) with a dimension no larger than 200X100 mm. In order to fit parts of any dimension onto the screen, a scaling factor is worked out. All calculations are performed in workpiece dimensions (parameters) and the scale factor is only used to convert the workpiece parameters into screen parameters before a line or a curve is plotted.

However, the position of the cursor on the workpiece is displayed in the text window in workpiece dimension. The cursors unit of movement is set to `0.2` or `0.5` mm depending on scale factors value. A background grid is provided to ease the design process and, depending on the workpiece dimensions, reference dots at `5` or `10` mm (in workpiece dimension) are spaced. The calculated scale factor is used to fit the workpiece onto the screen in at least on dimension. A border line is drawn to signify where the movement of the cursor (and subsequent plotting) would be valid for design purposes. A datafile, with the same name as the part-program, is created so that postprocess data can be written to. All these tasks are performed by `PROCinitialise`.

Users can then draw simple shapes by moving the cursor around the screen. Once the cursor has reached the desired position, a command from the text window may be entered to define what type of operation is to be performed. Movement of the cursor on the screen corresponds to the movement of the milling machines spindle, therefore it has to be told whether to cut the workpiece or fresh air, at what feedrate and depth of cut. As the cursor moves around the screen, its absolute position is updated constantly in the text window.

To send the cutting head from one point to another without it performing cutting, command `M` (`MOVE`) is entered once the cursor has reached the position which to the head is to be moved. `PROCmove` will position the cutting head above the workpiece before allowing it to move to a new position. A Straight cut can be carried out by entering command `L` (`LINE-DRAW`) once the cursor is situated at the end of the cutting position. `PROCline-draw` will plot a straight line between the two points on the screen.

Plotting a curve can be performed by positioning the cursor to a point where the curve should end, and entering command C (CURVE DRAW). This will produce the quadrant of a circle with the two point being the start and end points of the quadrant. In this way the user would not need to know the center point of the curve to be plotted. PROCcurve-draw will get the start and end position of the desired curve and then it works out the centre points for the quadrant. Letters 'C' and 'A' for clockwise and anticlockwise are printed on the graphic screen and it is required of user to choose whether a clockwise or anticlockwise quadrant should be plotted by entering 'C' or 'A'.

The feedrate of milling table can be changed by entering command F (FEEDRATE). For this option the cursor does not need to be moved. The feedrate is in mm/min and PROCfeedrate will require from user the new feedrate. Depth of cut can also be changed by entering command D (DEPTH) and it will get into effect in the next machining step. PROCdepth is used to perform this.

Design information is formatted (postprocessed) in such a way as to be understandable to the milling machines controller. Formatted data is stored in the form of data blocks [TERCO 1980] on to disc. This is performed every time a command is entered at the keyboard throughout the design process (except PRINT). If new values of parameters (e.g. depth of cut, feedrate etc.) are not entered into a data block, by default they are left the same as they were in the previous data block. Formatting of datablocks is performed by FNformat every time the program receives a command from the user (except PRINT). To download the part-program to the CNC-1000 the serial link (RS423) is used. Normally, BBC microcomputer formats data via ACIA chip [Dickens 1987] with no parity bit and a single stop bit. However, CNC-1000 requires the postprocess data (data blocks) to be sent with seven data bits, even parity and two stop bits. This is carried out whilst the data is being sent to the milling machines controller.

Removing of existing lines or curve is carried out by `PROCremove`. This is done by entering command `R` (`REMOVE`). The pointer of disc data file will be adjusted to the start of that data block so that the, next data block can be overwritten on that block. There exist several arrays which hold various design parameters. Information from these arrays are used to find out the types of previous command (e.g. whether the previous command was a line or curve). If the previous command is a line or a curve then a line or curve is plotted in the background colour from the present point.

A screen dump of part design can be obtained by entering command `P` (`PRINT`). `PROCprint` will first perform a screen dump, Fig. (B-3) and then gives a printout of the formatted design data, Table (B-1). To end the design session user will enter command `E` (`END`) as `PROCend` will close the datafile before ending the session.

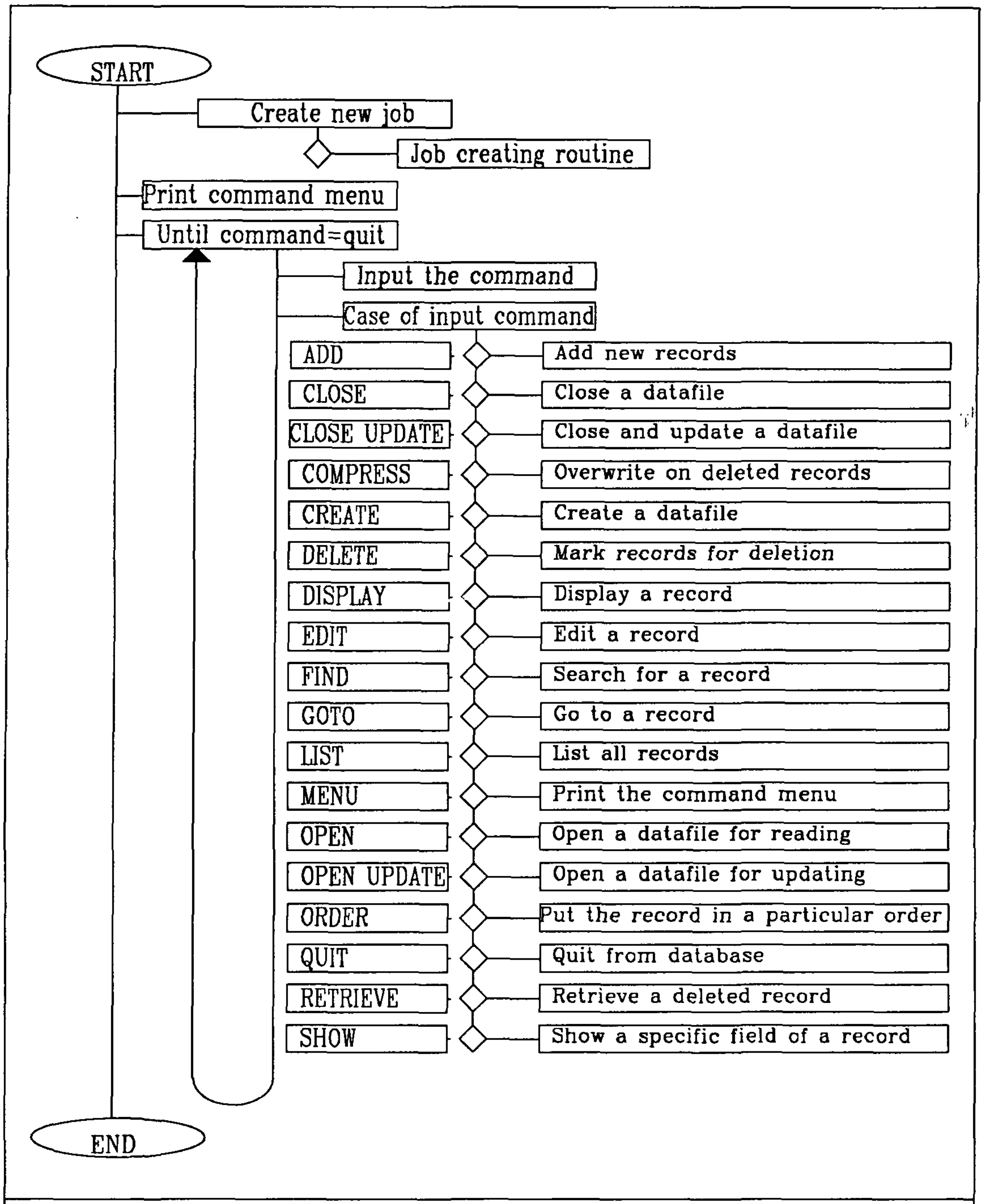


Figure (B–1) A flow chart for manufacturing database module.

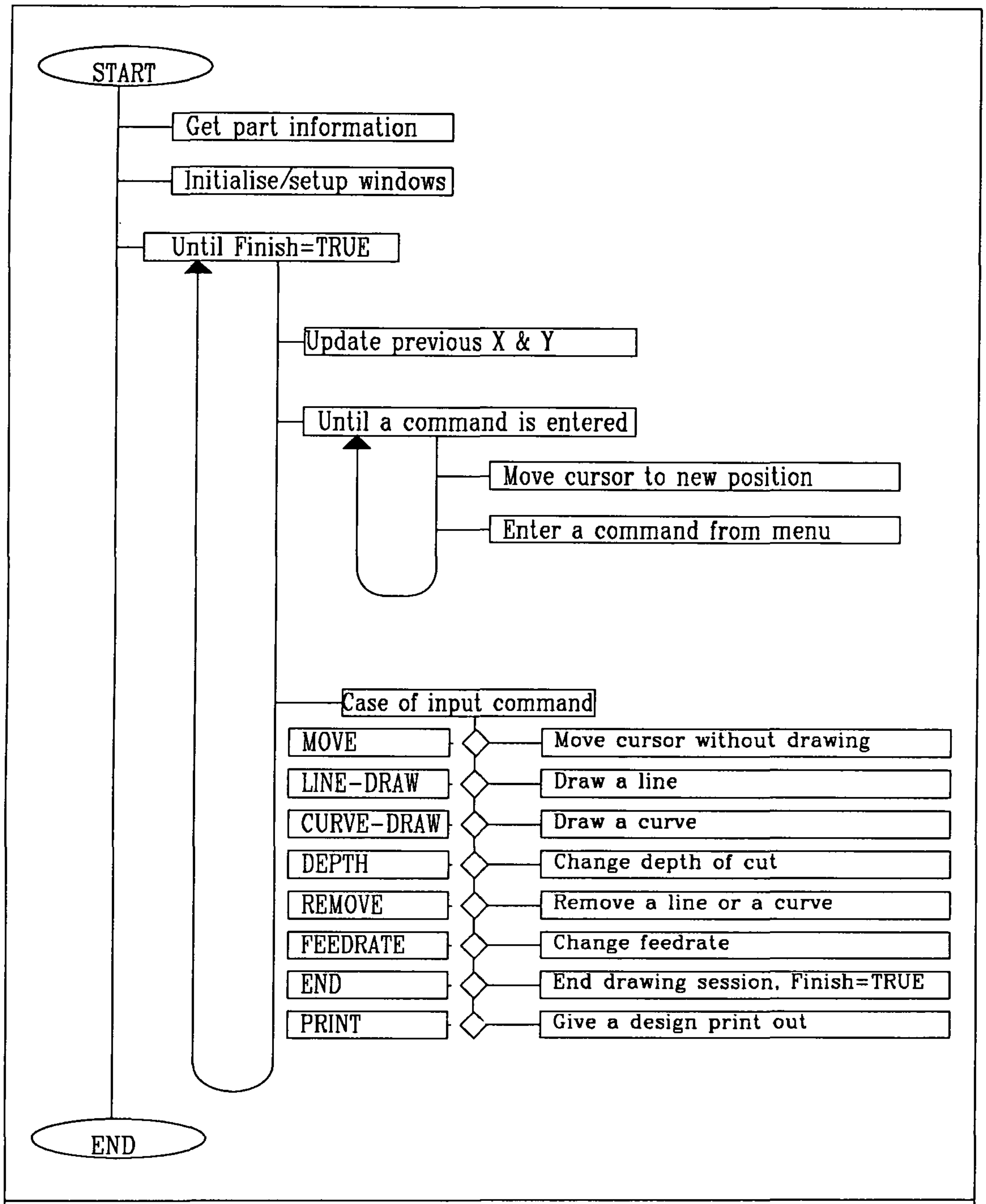
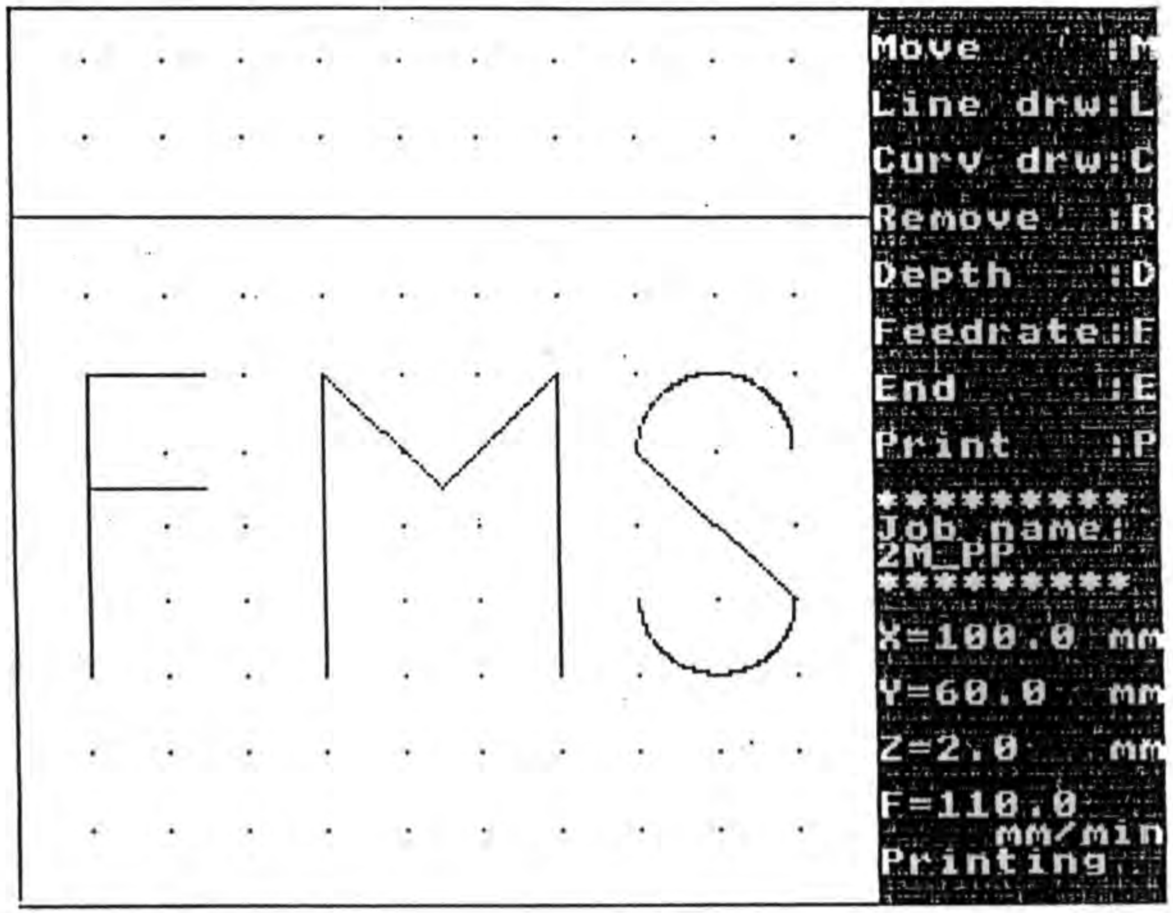


Figure (B-2) A flowchart for design for manufacture module.



```
*********
Job name:
2M_PP
*********
X=100.0 mm
Y=60.0  mm
Z=2.0   mm
F=110.0
    mm/min
Printing.
```

Figure (B-3) A part design screen dump produced by CAD10 program.

N	G	F	S	T	M	X	Y	Z	I	J	K
001	00			1							
002						02500	07000				
003		110	1		03						
004	01					01000		-00600			
005							03000				
006	00					02500	05500	00000			
007	01					01000		-00600			
008	00					04000	03000	00000			
009	01						07000	-00600			
010						05500	05500				
011						07000	07000				
012							03000				
013	00					08000	04000	00000			
014	01					09000	03000	-00600	00000	01000	
015	03					10000	04000		01000	00000	
016	01					08000	06000				
017	02					09000	07000		01000	00000	
018						10000	06000		00000	01000	
019	00				02						

Table (B-1) A post processed design data for the CNC-1000.

APPENDIX C

C.1 Inter-cell Real-time Scheduler

The overall scheduling of the FMS activities is carried out by a program called "SCHEDULER", Appendix F. Figure (C-1) shows a flow chart of the program. At the beginning of the program it is required of the user to enter the information regarding the particular job to be processed. The user, by inputting the name of the job, batch size for the job and test number on which a historical report can be referenced, invokes the operation of part production in the FMS. Next, the program searches through the manufacturing database to identify to which cells start-up messages should be sent.

The sequence in which the parts have to visit the cells is identified by the procedure `PROCseq-active-cell-check`. It checks whether or not the active cells are included in the part production sequence. Parts are then sent to the cells according to this predefined sequence. Procedures `PROCpart-trans-from-warehouse` and `PROCpart-trans-to-warehouse` will transfer raw parts from the warehouse to the first logical cell (i.e. the first cell in the part production route) and send a complete processed part to the finished parts store section of the warehouse respectively.

Monitoring of the production progress of the cells is partially carried out by `PROCoutput-partbuf-check`, as this procedure scans through different manufacturing cells to identify the one which has completed the operation on a part before depositing it to its output part buffer. If there already exists a part in the output part buffer then the program has to make enquires of the status of the input part buffer of the next logical cell before transferring the part to that cell. This is carried out by PROCinput-partbuf-check.

The order in which the production cells are checked is sequentially determined by the data contained within an index file. Should the situation arise that some cells have a higher importance than others, for example having very short operation time, then higher priority can be given to them by including that or those cell or cells more than once in the index file. This index file is created at the end of a job creation session. However, the speed at which the cells are checked by the program is far greater than the speed of the fastest operation that a cell could ever perform.

Other activities of production cells are monitored by System Monitoring Module (SMM), upon which real-time decisions are made. The integral part of SMM comprises two procedures namely PROCoutput-partbuf-check and PROCinput-partbuf-check. For example, SMM checks whether or not the cell has produced a defective part. If this is the case, procedure PROCdefective-part-trans-to-warehouse is called to ensure that the part is not transferred to the next logical cell but to the reject part store. SMM can be extended to monitor any number of cell critical status upon which more accurate decision may be based.

Transfer of parts from one cell to another is carried out by procedure PROCpart-trans-to-nextcell. Whenever a part is to be transferred to a cell or taken away from it, the scheduler will have to make the appropriate arrangements with that cell regarding its readiness for such an activity by exchanging messages between the two parties. Durham FMS utilises a pneumatic trolley (inter-cell part transfer mechanism) by which the part transfer between the cells is carried out. Currently, due to hardware problems the operation of the trolley is simulated by procedures PROCsend-trolley and PROCdrive-trolley but once it becomes operable, the control command can be included in these procedures.

Procedure `PROCterminate-part-prod` will terminate the process of part production in all of the cells by sending the appropriate messages to them. At the moment, once the job has been terminated it does not provide the option of continuing the job from where it was stopped. However, since all of the activities of the system have been recorded (explained in chapter six), the system has the potential to be developed further by utilising the information from the recorded data together with the current status of the production cells, for continuation of the terminated job.

Once the production of all of the parts has been completed successfully, the scheduler sends a message to each cell that the production period for the current job is over and this is performed by procedure `PROCproduction-period-comp`. Following this the scheduler passes its control to the `MASTER` program, where the upload file preparation is commenced. A report is required of all the production cells which have been currently involved in the processing of the job. This is discussed in chapter six.

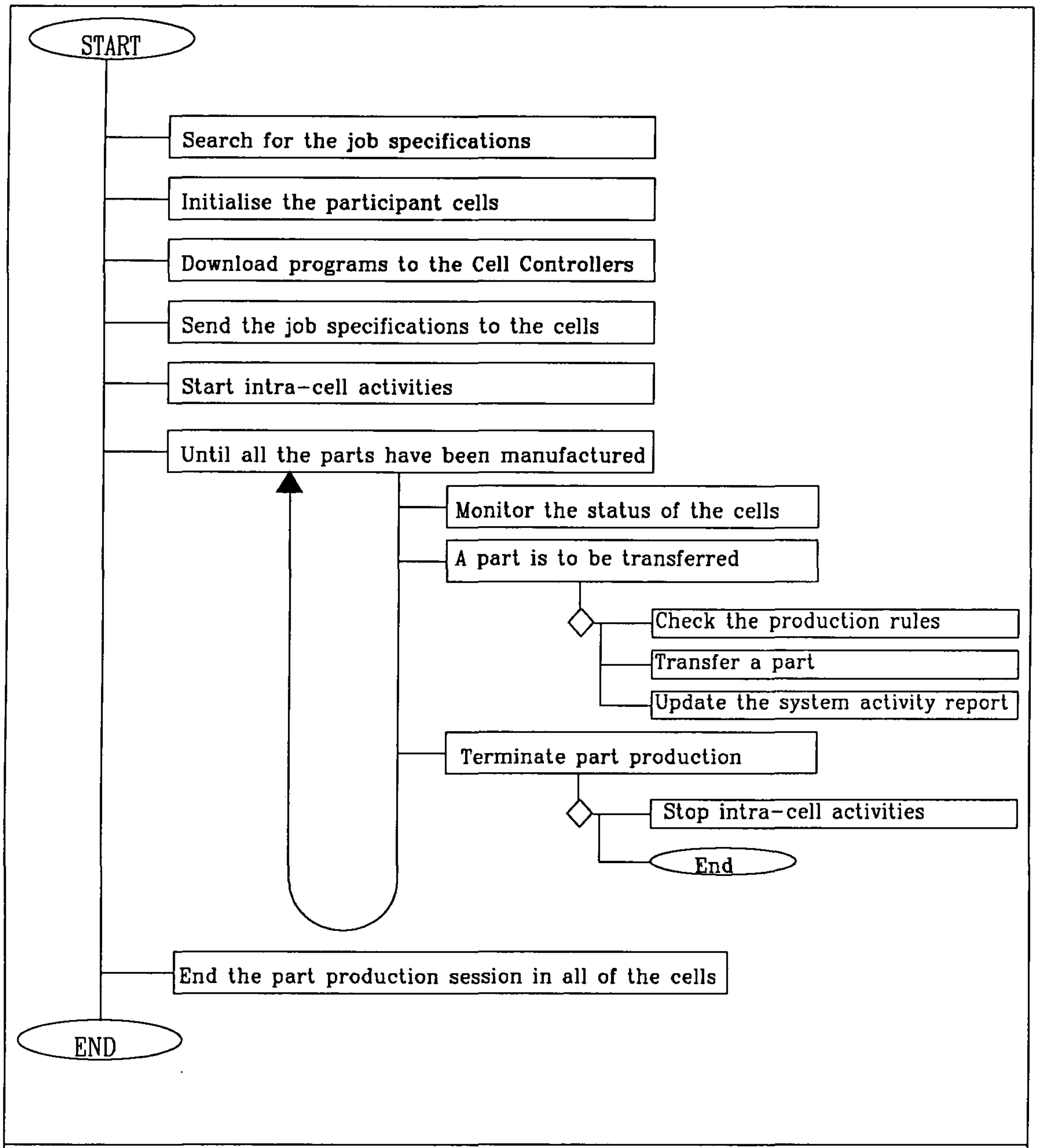

START
Search for the job specifications
Initialise the participant cells
Download programs to the Cell Controllers
Send the job specifications to the cells
Start intra-cell activities
Until all the parts have been manufactured
Monitor the status of the cells
A part is to be transferred
Check the production rules
Transfer a part
Update the system activity report
Terminate part production
Stop intra-cell activities
End
End the part production session in all of the cells
END

Figure (C-1) A flow chart for the FMS scheduler.

APPENDIX D

D.1 Cell Controller Specific Software (CCSS)

The general purpose cell controller software is in fact a CCSS program in which modification of the relevant section will result in a CCSS for the particular cell. Therefore, the functionality of the routine within the CCSS is the same for all cell controllers in the FMS and for this reason a CCSS program namely "MILL-VMD" belonging to the milling cell is chosen as an example for the description of cell controller module functionality, Appendix F.

Figure (D-1) shows a flow chart for the Cell Controller Specific Software. Once the CCSS program has been downloaded by the system scheduler into the memory of the milling cell controller, CCSS will utilise the manufacturing database to obtain information about the job. It will then initialise and download the appropriate programs to the controllers of the resources (such as robot and CNC milling machine) within the cell. Procedures PROCinitialise and PROCdownload perform this.

The program will continuously update cell status information in the dynamic database. It also checks whether a part is to be sent to the cell (by the system scheduler) or to be taken away from it. Transfer of a part to or from the cell involves the following cooperation of cell controller with the system scheduler. CCSS will receive a message from the scheduler that a part is going to be transferred to the cell. CCSS will then reply that it is ready for a part transfer to the cell. It then arranges with the cell's robot for the transfer of the part into the cell's input part buffer. Next, CCSS send control command to the robot to perform a pick and place operation, depositing the part onto the milling machine's table before issuing commands to the milling machine's controller to start its operation. Transferring of parts out of the cell takes a similar course. Procedures `PROCget-a-part` and `PROCsend-a-part` are used to transfer a part in or out of the cell. Also, `PROCstart-milling` will start the machining operation by sending the appropriate control command to the controller of the milling machine.

Once the milling operation is in progress, CCSS will regularly check the state of various sensors, from which a decision is made upon feedback signals which would represent the completion of the milling operation. The monitoring of the milling operation is carried out by a function `FNmill-monitor`. This routine has been coded in such a way that on certain occasion (for example if the part is the first to be processed in this cell it would be a defective part) to report that the machine has produced a defective part. This simple routine was used to see if the system has the capacity to cope with some of the problems which may arise in a real system (i.e. shop floor). As this will cause the system to increment the batch size by one for defective part replacement so that the initial demand for good parts (non-defective) is met.

There are four defined paths which the robot can take to perform a pick and place operation. These are; taking a part from the trolley (inter-cell part transferring mechanism) and placing it onto the input part buffer, taking a part from the input buffer and placing it onto the machine's table, taking the part from the machine's table and placing it into the output part buffer and finally, transferring a part from the output part buffer onto the trolley. Procedure `PROCpart-transfer` is used to execute this by receiving the information about the path before issuing commands to the robot to perform the desired pick and place operation. Due to both electrical and mechanical problems with the robots their operations were simulated in all of the cells. However, if a cell is equipped with a working robot, the control signals must to be included where the simulated routines are placed.

Procedure `PROCprocess-next-part` is used only when the operation on a part has been completed and the input part buffer is not empty. The situation may arise where the processing time on a part for the current cell is far less than for the next cell. The output part buffer may fill up (if the size of buffers are small) too quickly. Procedure `PROCout-partbuf-full` is then used to first empty at least one part before placing the current processed part from machine's table into the output part buffer.

In order to demonstrate that the production cells can negotiate about part processing (exhibiting the heterarchical behaviour) with each other without the intervention of the system scheduler, residing at the master level, before a part is transferred to a cell, the two cells will first exchange information about that part before transferring it. For example, parts leaving a FMS cell may be placed on a pallet; the location and orientation of each part being placed onto the pallet by the previous cell's robot is then required by the receiving cell. To obtain this information from one another, cells can utilise the `CellTalk` (explained in chapter five). This is how a FMS cell utilises the communication module for the exchange of information between the cells. Quality control could also be exercised by the manufacturing cells by each cell assigning a number for the quality control parameter at the end of processing period of a part in that cell. Depending on how well and accurately the processes are performed in each cell the cumulative number for this parameter could decide at what stage of part production through the different manufacturing cells a part is categorised as defective hence further operation on that part to be abundant.

The example of this feature in the Durham FMS is shown where, if a part is to be transferred from the milling cell to the assembly cell, the assembly cell will first make enquires about that part of the milling cell and, once it has received the information (e.g. part orientation or dimension etc.) from the milling cell, it will then accept the part into its cell. The procedures which deals with this are `PROCinfo-request`, `PROCrespond`, `PROCget-mail`, `PROCmail-test`, `PROCtx-message` (send mail), `PROCpeek` and `PROCpoke` which are explained in the manufacturing communication section of chapter five.

The two procedures `PROCget-part-orientation` and `PROCget-part-dimension` are used to exhibit the direct information exchange of two production cell during the period of part production in real-time. They simulate the functions of systems such as a vision system or coordinate measurement system by simply generating a string composed of a series of characters and numbers. These characters and numbers have no real meaning and are used only to highlight the exchange of information between the entities.

To demonstrate the hierarchical behaviour of the system a routine is built into the CCSS to receive commands from the system scheduler (residing at the master level) for the termination of part production in that cell. Once this command is received by the CCSS it will terminate the part processing if and only if the current operation is completed. It then produces a report representing the state of the cell at the time of termination. Procedures `PROCterminate-part-prod` and `PROCcell-reports` will carry out this.

Finally, once the CCSS receives the commands from the system scheduler which indicate that the desired goal (production of the batch) has been achieved, the cell controller produces a report on the production of parts before uploading it to the MASTER program (residing at the master level). Due to the restrictions imposed by the available memory of the BBC microcomputer to compute and produce a meaningful report for the CCSS, a simulated report (i.e. concocted information) on the cell performance during the production period demonstrates the uploading capability of the cells. Such a report may include information such as the cell efficiency, its defect rate, number of processed parts in that cell etc. Once the report is uploaded it becomes resident in the records of the historical report data on the production cells.

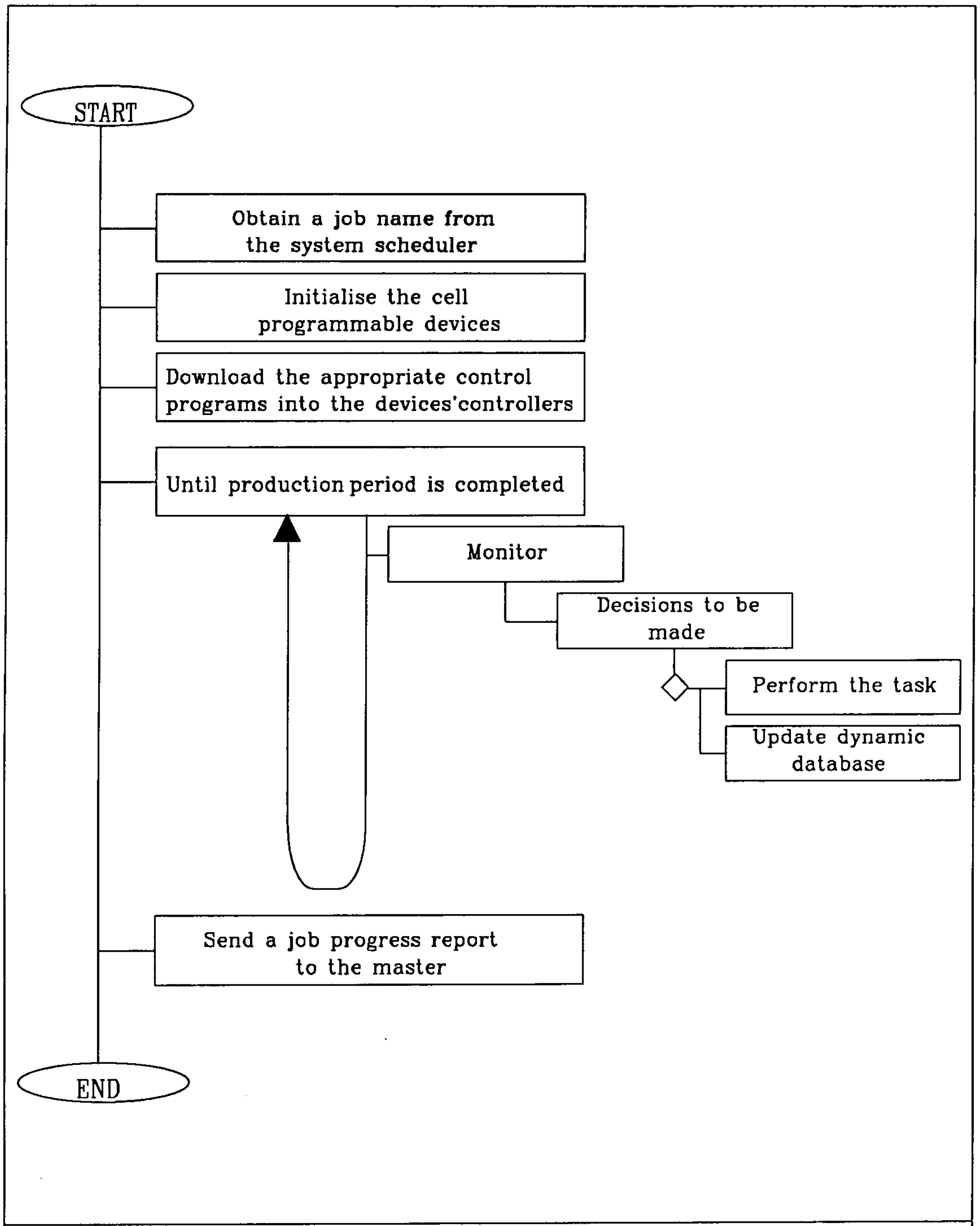

START
Obtain a job name from the system scheduler
Initialise the cell programmable devices
Download the appropriate control programs into the devices'controllers
Until production period is completed
Monitor
Decisions to be made
Perform the task
Update dynamic database
Send a job progress report to the master
END

Figure (D-1) A flowchart for Cell Controller Specific Software.

APPENDIX E

The Seventh Layer of the OSI model

E.1 Application Layer

The application layer is the top layer of the OSI reference model and sometimes is referred to as Application Entity (AE). The application layer provides services to allow application programs (user programs) to access the network and it also deals with semantics (meaning) of information exchange between application processes [Mcconnel 1988]. The user programs are normally referred to as Application Process (AP).

Within the application layer, Fig. (E-1), two divisions may be defined : Common Application Service Elements (CASE) and Specific Application Service Elements (SASEs).

SASEs have been developed to allow APs to access and provide specific services. File Transfer, Access and Management (FTAM), Manufacturing Message Services (MMS) and Directory Services (DS) are some examples of SASEs . CASE provides some general supporting services to which most APs will require access. The Association Control Service Elements (ACSE) is an example of a CASE which is concerned with the setting up and closing down logical association (connection) between two correspondent SASEs. Since in manufacturing MAP uses the FTAM and MMS, the following brief description of both was considered appropriate.

E.1.1 File Transfer, Access and Management (FTAM)

The File Transfer, Access and Management (FTAM) is a SASE which provides the means of remote access and management to files stored on different computers (maybe file servers), irrespective of the type of networks to which they are attached. Consider two computers; one as a client on the broadband factory backbone and the other a server on the baseband CSMA/CD network, Fig. E-2. The client (initiator) may wish to operate on a file which resides or is to be created in the server's (responder) file store. This commences once the application process of the client requests its FTAM of Application Entity (AE) to operate on a file in the server. The request is transmitted by the network to the FTAM of AE of the server. The server's FTAM then accepts the request and carries out the appropriate file operations.

Nine functional units are defined by FTAM, each associated with a set of file services such as read, write, restart data transfer. FTAM guarantees to read or change file attributes, create, delete or transfer files, erase file contents, locate specific records and read and write records of a file.

E.1.2 Manufacturing Message Service (MMS)

The Manufacturing Message Service (MMS) is a SASE which has been developed for exchanging messages between manufacturing shop floor computer-based equipment such as robot, NC machine, cell controller, etc. FTAM may transfer files to the cell controller of each manufacturing cell connected on the factory backbone (MAP broadband) but the transfer of files within each cell is utilised by services offered by MMS (i.e. file transfer).

One of the ways which MMS can load user programs into shop floor programmable devices is as follows; a cell controller uses MMS to send a message to the device controller, (which must be MAP compatible), telling it to request a program and leaves it to get on with locating the program in a remote database, file store or its own memory and then run that program. In this way, the cell controller does not need to know about the programmable device which it is controlling, hence, vendor independence is achieved. The other advantage would be in the program loading process in which either the whole program or part of it at a time, (so-called drip feed), can be loaded. This eliminates the constraint imposed by the size of the device controller's own memory, opening doors to more complicated operations, such as in-process monitoring, etc. It can be said that MMS [Williams 1988] is concerned with the communication and interworking of computer-based equipment on the manufacturing shop floor and is the key to achieving vendor independent interoperability between shop floor devices.

MMS provides a standard syntax for messages between shop floor devices. It assembles frames of data into a complete message and defines the message notation. The information contained in the message notation is that about the purpose, length and contents of each element in a message. MMS uses Abstract Syntax Notation one (ASN.1) to unambiguously define the content of a Protocol Data Unit (PDU). MMS also uses the concept of client and server, and the services of the protocol are modelled by the interaction of two Manufacturing Message Protocol Machines (MMPMs), Fig. (E-3). The MMPM interfaces to OSI from ACSE and presentation layer. It is independent of type of network and provides services to its user. MMS user represents the application process (application program), uses the services provided [BITS 704 1989] by MMPM, and communicates with the peer entity on the network by exchanging PDUs [BITS 705 1989]. MMS has four types of service primitives: request, indication, response and confirm. The exchange of information between the client and server takes place in the following way: client (MMS user) sends a service request to its MMPM and as a consequence of this MMPM provides a PDU for transmission to its peer entity in the server (MMS user), Fig. (E-4). If the service request is successful the confirm or response primitives are positive and if unsuccessful the aforesaid primitives are negative. This means that if MMPM in the server receives a response positive primitive from its MMS user it will then transmit a response PDU. Should MMPM receive a response negative primitive it will transmit an error PDU.

There are two types of MMS services. In confirmed MMS services all four service primitives, (request, indication, response and confirmation), are present and where the response and confirmation service primitives are absent they are called unconfirmed MMS services.

MMS is a generic standard, (sometimes referred to as MMS core), for any device on the factory floor and it takes the form of a basic abstraction without making any reference to specific devices such as robots, PLCs etc.

The Companion Standard (CS) will establish the relationship between the generic MMS and the specific requirements of a particular shop floor device and also describe the effect that the generic MMS services have on shop floor devices. For example, a named variable N.FRL represents a feed rate limit value in a NC machine and writing to named variable N.FRO applies a feed rate override. There exists one companion standard for each application area such as robot, NC, PLC etc. [BITS 695 1989], [BITS 696 1989], [BITS 706 1989].

The companion standards allow the MMS services and protocol to be extended to further refinement, (if necessary), of their definition for a specific task. For example, in the PLC companion standard, under the stop semantic, there are some additional parameters to represent the exact meaning of stop, whether it be stop right now, end of the cycle or the end of the operation and this parameter does not exist in MMS abstract syntax. Should there be a need for any changes, modification can be done in the companion standard rather than in the MMS, leaving the MMS standard to remain in its stable state.

The idea of a Virtual Manufacturing Device (VMD) has been introduced in RS 511 draft 6. It models the MMS server and represents its functionality and resources. Figure (E-5) shows the logical position of the VMD within a MMS client/server model. There exists a set of abstract objects at the VMD (server) by which they are modified.

An object can be a domain and in the case of a robot the domain might be that portion which deals with motion control for some or all parts of the robot arm. The behaviour of the VMD is described by: defining the objects, defining the operations which may be performed on them and describing the object relationship to the real device. A client may remotely manipulate the objects in a server and the server's application tracks the inherent real resource to match with the objects.

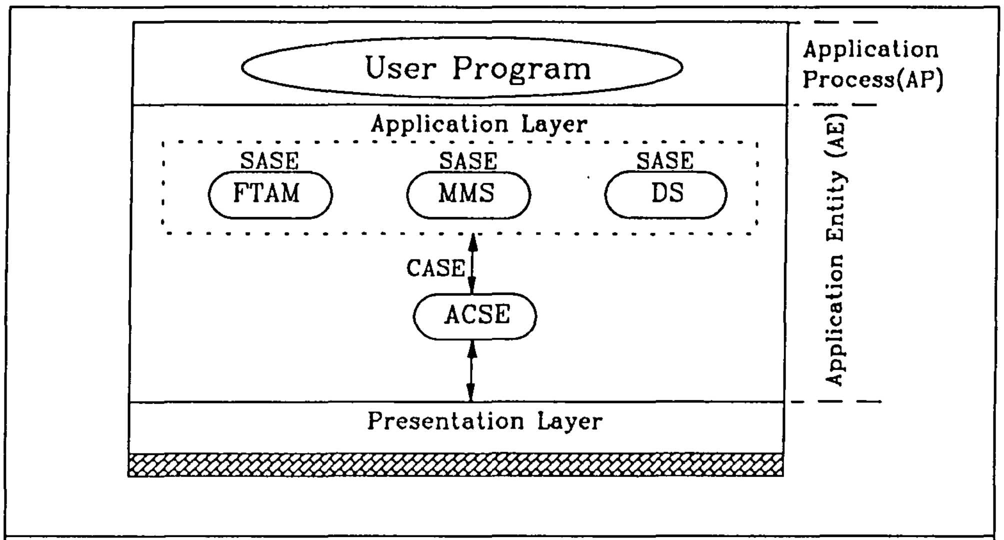


Figure (E-1) Logical position of SASE and CASE in the Application layer.

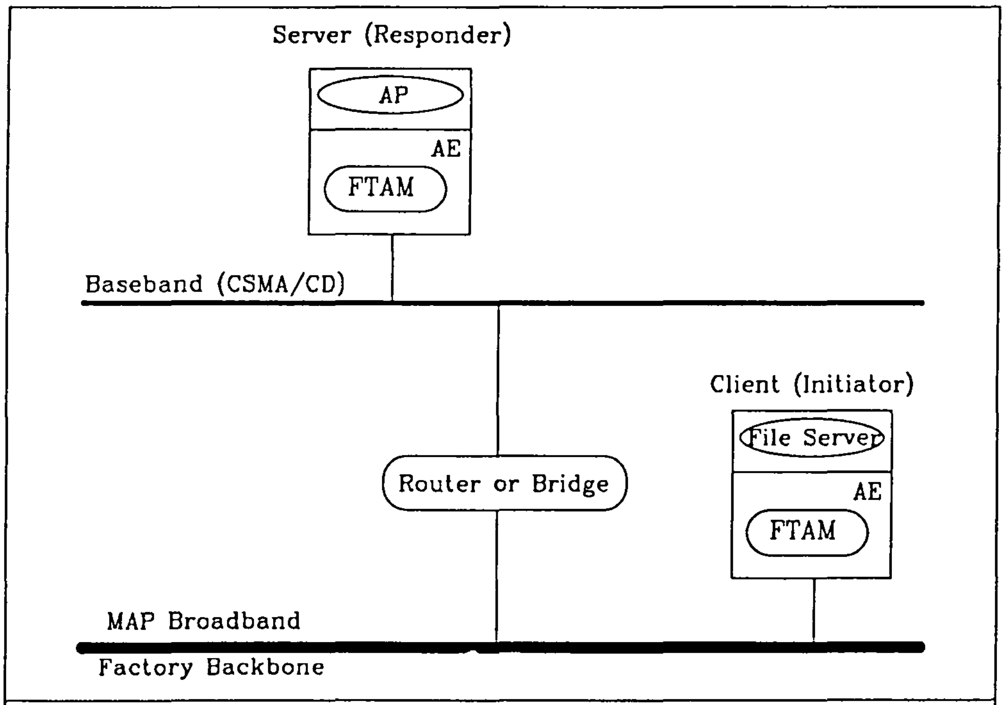


Figure (E-2) File Transfer Access and Management between client and server.

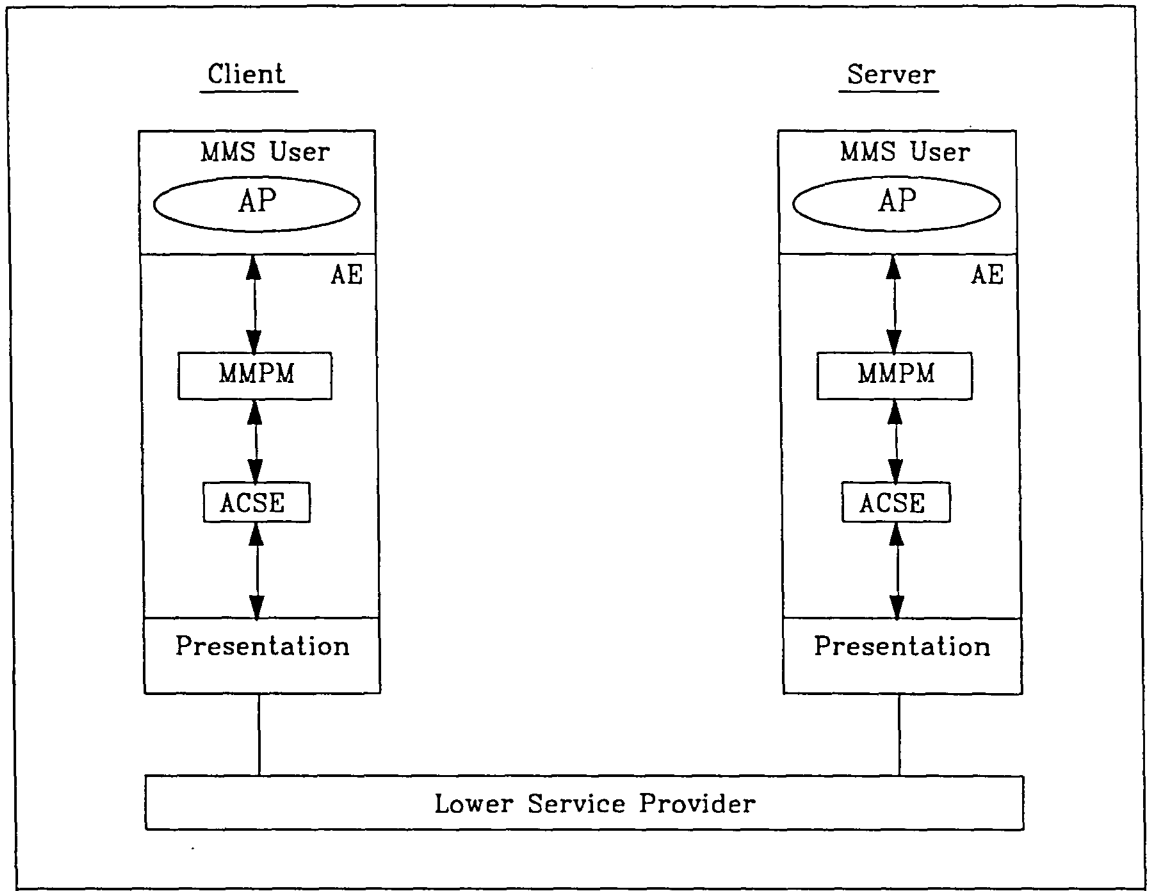


Figure (E-3) Client / Server and Manufacturing Message Protocol Machines (MMPM) interaction.

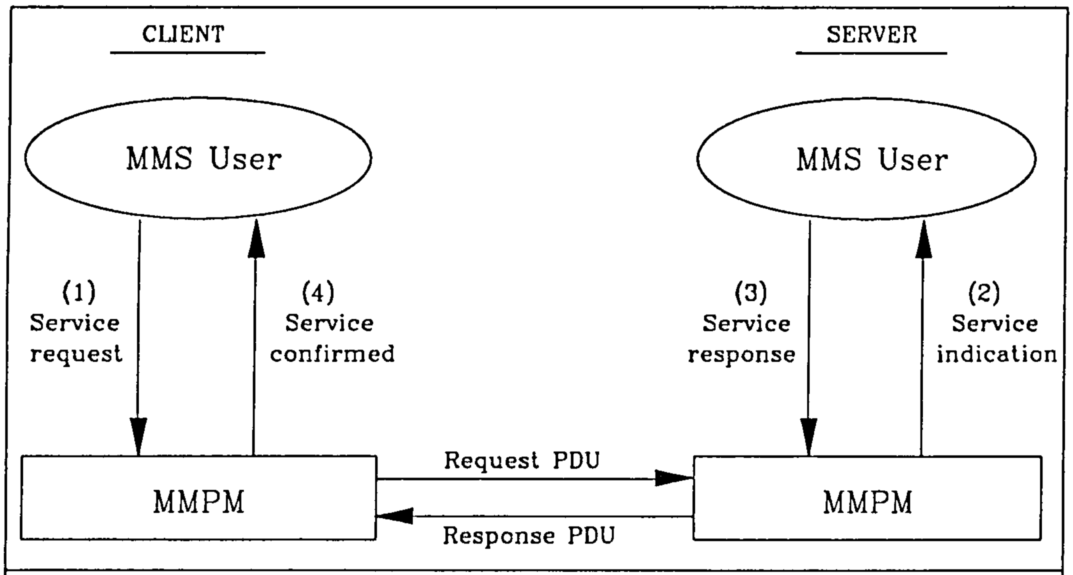


Figure (E-4) Confirmed MMS services.

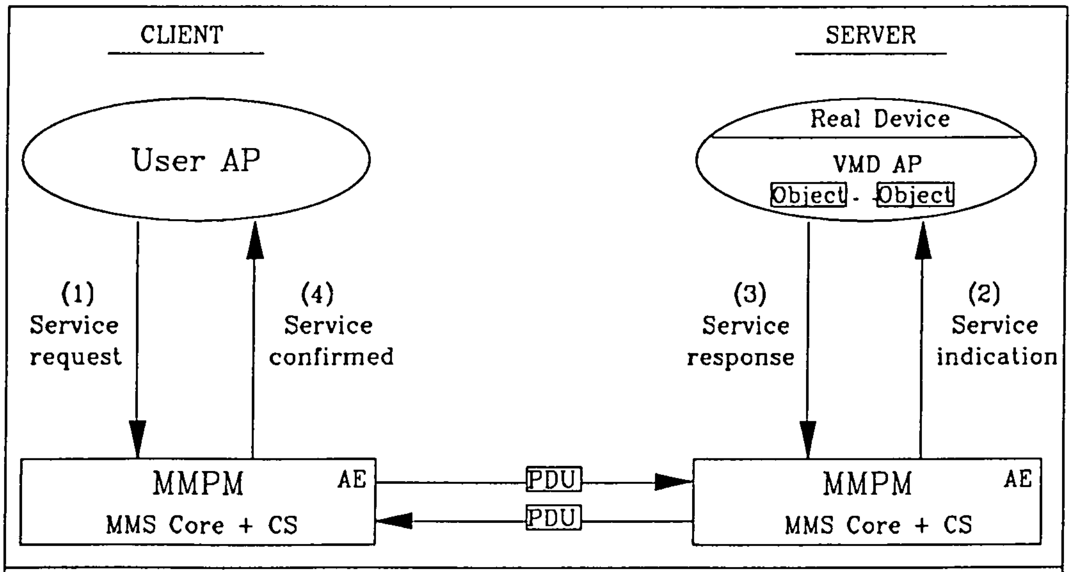


Figure (E-5) Logical position of VMD within the client / server interaction.

APPENDIX F

The listing of programs

In this section a listing of the most important routines utilised for the Durham FMS are presented in alphabetical order.

```
L.
   10REM...........................ACT_CHART.................................
   20REM This program reads the recorded data on the system activities and represents them graph
ically.
   30MODE 7
   40REM Initialise various variables.
   50op_TYPE=1:key1=op_TYPE:CC_NAME=2:key2=CC_NAME:CC_CON=3:key3=CC_CON:STAT_NO=4:key4=STAT_NO:N
OF_PP=5:key5=NOF_PP:PP_NAME=6:key6=PP_NAME:SEQ_NO=7:key7=SEQ_NO
   60REM Define the number of records which are resident in the current data file so that an ind
ex array can be set up.
   70CLS:max_recs=6:REM Maximum number of records, a record for each cell.
   80REM Set the number of fields per record.
   90rec_fields=12
  100DIM field_cont$(rec_fields),index$(max_recs,2),field$(rec_fields),type$(rec_fields),width$(
rec_fields),dp$(rec_fields),command_stack$(8)
  110quit=FALSE:CLS
  120CLS:PRINT TAB(3,10):INPUT"Enter the name of an existing executed job :"job_name$
  130PRINT TAB(3,13):INPUT"Do you require a hard copy of the activity sequence chart (Y/N) ";p$
  140PRINT TAB(3,16):INPUT"Enter the test number :"testno$
  150PRINT TAB(0,23) STRING$(37," "):PRINT TAB(1,23) "Wait..."
  160REM Encode the job name to obtain the relevant data file in which the activity data is stor
ed.
  170job_timing$=job_name$+"_A"+testno$
  180command_stack$(2)=job_name$:command_stack$(3)="INDEX":command_stack$(4)=job_name$+"_INS":PR
OCopen_file
  190nofcell%=0:screen%=1
  200FOR I=1 TO VAL(nofrec$)
  210IF index$(I,1)<>"00" THEN nofcell%=nofcell%+1
  220NEXT I
  230act_ch=OPENIN(job_timing$)
  240INPUT£act_ch,batch_size$
  250nofcell%=nofcell%+3
  260y_scale=900 DIV (nofcell%)
  270page_num=0:interval=0
  280x_off_set=20:y_off_set=45
  290last_xpos%=x_off_set:last_ypos%=y_off_set
  300last_char$="N":last_cell%=2:first_act%=0
  310DIM cell%(y_scale)
  320MODE0
  330PROCaxis
  340REPEAT
  350PROCplot_timing(p$)
  360UNTIL EOF£act_ch
  370g=GET
  380CLOSE£act_ch
  390command_stack$(2)=job_name$:PROCclose_file
  400MODE 7
  410REM Go back to the main program.
```

```
420CHAIN"MASTER"
430END
440REM
450
460REM The Manufacturing database resides here.
470
480
490DEF PROCaxis
500REM Set up a window to represent the active cell for the current job.
510VDU19,1,4,0,0,0:VDU19,0,2,0,0,0
520REMGCOL 0,1:GCOL 0,128:COLOUR 0:COLOUR 129
530REMVDU19,0,1,0,0,0:VDU19,1,4,0,0,0
540REMx_off_set=20:y_off_set=35
550last_pos%=0
560page_num=page_num+1
570MOVE x_off_set,y_off_set
580DRAW 1280,y_off_set
590MOVE x_off_set,y_off_set
600DRAW x_off_set,900
610MOVE x_off_set,900
620DRAW 1280,900
630FOR I=1 TO nofcell%
640MOVE x_off_set,I*y_scale
650DRAW 1280,I*y_scale
660cell%(I)=(I+1)*y_scale
670NEXT I
680VDU5
690half=y_scale DIV 2
700MOVE 1264,half+y_off_set-8
710PRINT"T"
720MOVE 400,28
730PRINT"System_wide Scheduling Chart"
740MOVE 1250,57
750PRINT" >"
760MOVE 1250,y_scale+12
770PRINT" >"
780MOVE 0,half+30
790IF page_num=1 THEN PRINT"O"
800MOVE x_off_set,1015
810PRINT"R:Warehouse   ";"F:Finished/Reject parts store";"     **";job_name$;"**   ";"**Batch
size=";batch_size$;"**"
820MOVE x_off_set,978:PRINT"M:Milling  ";"S:Sawing  ";"L:Turning  ";"V:Vision  ";"W:Washing ";
" <cell>";"    **Test no.";testno$;"**"
830MOVE x_off_set,941:PRINT"D:Defective  ";"G:Good  ";"<parts> ";"  T:Activity conclusion time
(Min)";"  **screen ";STR$(screen%);"**"
840screen%=screen%+1
850MOVE 0,y_off_set+y_scale+half-30:PRINT"R"
860MOVE x_off_set-7,880:PRINT CHR$(94)
870MOVE x_off_set-7,860:PRINT CHR$(94)
880MOVE 0,(nofcell%*y_scale)-y_scale+half:PRINT"F"
890FOR I=1 TO VAL(nofrec$)
900IF index$(I,1)<>"00" THEN cell$=LEFT$(FNget_field(VAL(index$(I,2)),key1),1):y_pos=VAL(index
$(I,1)):MOVE 0,(y_scale)*(y_pos+1)+half
910IF index$(I,1)<>"00" THEN IF cell$="T" THEN PRINT "L"
920IF index$(I,1)<>"00" THEN IF cell$<>"T" THEN PRINT cell$
930NEXT I
940ENDPROC
950
960
970DEF PROCplot_timing(print$)
980REM Write the conclusion time of each activity at the bottom of the chart.
```

```
  990screen_lim%=0
 1000last_xpos%=x_off_set
 1010REPEAT
 1020INPUT£act_ch,activity$
 1030y_pos=VAL(activity$)+1:this_cell=y_pos
 1040char$=MID$(activity$,5,1)
 1050time$=MID$(activity$,6)
 1060interval=interval+1
 1070IF char$="G" OR char$="D" THEN ypos%=(nofcell%*y_scale)-half+5:MOVE last_xpos%+x_off_set-16
,ypos%+50:PRINT char$:GOTO 1090
 1080ypos%=cell%(y_pos)-half
 1090MOVE last_xpos%,last_ypos%
 1100IF last_char$="R" OR last_char$="F" THEN MOVE last_xpos%,last_ypos%:PLOT 5,last_xpos%+x_off
_set-10,ypos%-10:PROCarrow_up_f(last_xpos%+x_off_set-18,ypos%-15)
 1110IF first_act%<>1 THEN GOTO 1130
 1120IF last_char$<>"R" OR last_char$<>"F" THEN PLOT 21,last_xpos%+x_off_set-10,ypos%-15
 1130first_act%=1
 1140IF last_char$="T" OR last_char$="G" OR last_char$="D" THEN IF this_cell<last_cell% THEN PRO
Carrow_down_e(last_xpos%+x_off_set-2,ypos%+5)
 1150IF last_char$="G" OR last_char$="D" THEN PROCarrow_down_e(last_xpos%+x_off_set-2,ypos%+5)
 1160IF last_char$="T" THEN IF this_cell>last_cell% THEN PROCarrow_up_e(last_xpos%+x_off_set-2,y
pos%-15)
 1170IF interval=2 THEN MOVE last_xpos%,y_off_set+y_scale-30:PRINT CHR$(108):MOVE last_xpos%,y_o
ff_set+y_scale-63:PRINT time$
 1180IF interval=4 THEN MOVE last_xpos%,y_off_set+17:PRINT CHR$(108):MOVE last_xpos%,94:PRINT ti
me$:interval=0
 1190last_xpos%=last_xpos%+60:last_ypos%=ypos%-15
 1200IF char$="T" OR char$="F" OR char$="D" OR char$="G" OR char$="R" THEN last_xpos%=last_xpos%
-30
 1210screen_lim%=screen_lim%+1
 1220last_char$=char$:last_cell%=this_cell
 1230UNTIL screen_lim%=40 OR EOF£act_ch
 1240IF EOF£act_ch THEN IF print$<>"Y" AND print$<>"y" THEN ENDPROC
 1250IF print$<>"Y" AND print$<>"y" THEN GOTO 1270
 1260*SCDUMP
 1270g=GET
 1280CLG
 1290IF EOF£act_ch THEN ENDPROC
 1300PROCaxis
 1310ENDPROC
 1320
 1330
 1340DEF PROCdump
 1350REM Dump the image of the current screen onto the attached printing device.
 1360VDU2
 1370VDU1,27
 1380VDU1,65
 1390VDU1,8
 1400FOR J%=0 TO 80
 1410VDU1,27
 1420VDU1,42
 1430VDU1,5
 1440VDU1,512 MOD 256
 1450VDU1,512 DIV 256
 1460BYTE%=&7D87 + J%*8
 1470FOR SCAN1%=0 TO 31
 1480FOR I%=0 TO 7
 1490VDU1,(?(BYTE%-I%))
 1500VDU1,(?(BYTE%-I%))
 1510NEXT
 1520BYTE%=BYTE%-640
```

```
1530NEXT
1540VDU1,13
1550NEXT
1560VDU1,12
1570VDU1,27
1580VDU3
1590ENDPROC
1600
1610
1620DEF PROCarrow_up_f(abs_x,abs_y)
1630REM Draw an upward solid arrow.
1640MOVE abs_x,abs_y
1650PLOT 81,16,0
1660PLOT 81,-8,30
1670PLOT 81,-8,-30
1680ENDPROC
1690
1700
1710DEF PROCarrow_down_e(abs_x,abs_y)
1720REM Draw a downward dotted arrow.
1730MOVE abs_x,abs_y+8
1740PLOT 1,-8,-30
1750PLOT 1,-8,30
1760ENDPROC
1770
1780
1790DEF PROCarrow_up_e(abs_x,abs_y)
1800MOVE abs_x,abs_y-20
1810REM Draw an upward dotted arrow.
1820PLOT 1,-8,30
1830PLOT 1,-8,-30
1840ENDPROC
>
```

```
L.
   10REM.............................. BOARD_TEST .............................
   20REM This program generates the required signal for page select FRED and assigns prot B as o
utput on VIA£1. This port is chosen as an example to show the testing procedure for the board.
   30REM Check pin no.23 and 24 if VIA£1 is selected then these pins should go low and high resp
ectively.
   40REM To check whether the board is operating correctly, this program loads one (decimal) ont
o the selected output stream and shift that bit to the left once and then outputs it again, it r
epeats this eight times.
   50REM One of the oscilloscope probesshould be sunk on the least significant bit whilst the ot
her one is connected to the other bit in ascending order, a shift of one bit on every pin must b
e observed, or else the circuit is not working.
   60OSBYTE=&FFF4
   70FOR opt%=0 TO 3 STEP 2
   80P%=&2000
   90[
  100OPT opt%
  110 .INIT LDA £&93\ OSBYTE to write to fred.
  120       LDX £&E2\ Offset DDRB.
  130       LDY £&FF\ Value to be written.(all ports as output.)
  140        JSR OSBYTE
  150        LDA £&93
  160        LDX £&EB\Set ACR to zero
  170        LDY £0
  180        JSR OSBYTE
  190        LDA £&93
  200        LDX £&EE\Set IER to zero
  210        LDY £0
  220        JSR OSBYTE
  230       LDA £1
  240       STA &80
  250       RTS
  260]
  270[
  280 OPT opt%
  290 .LOOP1 CLC
  300 .LOOP2 LDA £&93
  310       LDX £&E0\output to port_B.
  320       LDY &80
  330       JSR OSBYTE
  340       ASL &80
  350       BCC LOOP2
  360        LDX £&F
  370 .LOOP3 DEX
  380        BNE LOOP3
  390       LDA£1
  400       STA &80
  410       JMP LOOP1
  420       RTS
  430]
  440NEXT opt%
  450CALL INIT
  460CALL LOOP1
  470END
>L.
   10REM..............................BOARD_TEST1 ................................
   20REM This program generates the required signal for page select FRED and assigns prot B as i
nput on VIA£1.
   30REM Check pin no.23 if VIA£1 is selected then this pin should go low.
   40REM Check the RS3_0 pins 38 to 35 for DDRB, the state of these pins should be L_L_H_L.
   50OSBYTE=&FFF4
   60FOR opt%=0 TO 3 STEP 2
   70P%=&2000
   80[
   90OPT opt%
  100 .INIT LDA £&93\ OSBYTE to write to fred.
  110       LDX £&E2\ Offset DDRB.
  120       LDY £0\ Value to be written.(all ports as input.)
  130       JSR OSBYTE
  140       LDA £&92
  150       LDX £&E0
  160       JSR OSBYTE
  170       TYA
  180       SEC
  190       SBC £28
  200       STA &0080
  210       RTS
  220]
  230NEXT opt%
  240REPEAT
  250CALL INIT
  260G%=?&0080
  270REM An absolute encoder is used as a device to provide the input signals to the port. Shoul
d the board fail at any instant a default value of 255 will be read for the current position of
the encoder.
  280REM The position of the encoder is set to a value other than 255 at the start of the test.
  290PRINT TAB(2,19);"Position in decimal="G%,
  300PRINT TAB(2,20);"Position in hex    ="~G%,
  310UNTIL FALSE
>
```

L.

```
   10REM ...............................CAD10....................................
   20REM This is a drawing program which is used to design simple shapes.
   30MODE 7
   40REM Set up the variables.
   50xpos%=0:ypos%=0:line_draw=0:move=1:curve_draw=2:depth_change=3
   60x_movement=0:y_movement=0:depth=0
   70x$="":y$="":z$="":f$="":n$=CHR$(&0A)
   80DIM command_stack%(6),data_block_len%(6),privious_xmovement(6),privious_ymovement(6)
   90PROCpart_info
  100PROCinitialize
  110finished=FALSE
  120@%=&20109
  130*FX4,1
  140PROCtext_screen
  150ON ERROR OFF
  160*FX15,0
  170REPEAT
  180REM Move the cursor around the screen and enter a command.
  190last_xpos%=privious_xmovement(1)*scale_factor:last_ypos%=privious_ymovement(1)*scale_factor

  200REPEAT
  210REM Display the cursor movement as it is manouvered around the screen.
  220PLOT 69,xpos%,ypos%
  230K=INKEY(0)
  240IF K=136 THEN x_movement=x_movement-unit_movement
  250IF K=137 THEN x_movement=x_movement+unit_movement
  260IF K=138 THEN y_movement=y_movement-unit_movement
  270IF K=139 THEN y_movement=y_movement+unit_movement
  280REM Update the value of the cursor position.
  290PRINT TAB(0,22)"X=";x_movement;TAB(8,22)"mm"
  300PRINT TAB(0,24)"Y=";y_movement;TAB(8,24)"mm"
  310PRINT TAB(0,26)"Z=";depth;TAB(8,26)"mm"
  320PRINT TAB(0,28)"F=";feedrate%
  330PLOT69,last_xpos%,last_ypos%
  340PLOT71,xpos%,ypos%
  350xpos%=x_movement*scale_factor:ypos%=y_movement*scale_factor
  360UNTIL K=&4D OR K=&4C OR K=&43 OR K=&52 OR K=&44 OR K=&46 OR K=&45 OR K=&50
  370REM Depending on the selected command from the menu, call the appropriate procedure.
  380IF K=&4D THEN PROCmove(x_movement,y_movement,scale_factor)
  390IF K=&4C THEN PROCline_draw(x_movement,y_movement,scale_factor)
  400IF K=&43 THEN PROCcurve_draw(privious_xmovement(1),privious_ymovement(1),x_movement,y_movem
ent,scale_factor)
  410IF K=&44 THEN PROCdepth
  420IF K=&52 THEN PROCremove(privious_xmovement(2),privious_ymovement(2),privious_xmovement(1),
privious_ymovement(1),scale_factor)
  430IF K=&46 THEN PROCfeedrate
  440IF K=&45 THEN PROCend
  450IF K=&50 THEN PROCprint
  460UNTIL finished
  470END
  480
  490
  500
  510DEF PROCassen_var_update(x_val,y_val)
  520REM Update the variables of the previous six commands.
  530LOCAL I
  540I=6
  550REPEAT
  560command_stack%(I)=command_stack%(I-1)
  570data_block_len%(I)=data_block_len%(I-1)
```

```
580privious_xmovement(I)=privious_xmovement(I-1)
590privious_ymovement(I)=privious_ymovement(I-1)
600I=I-1
610UNTIL I=1
620privious_xmovement(1)=x_val:privious_ymovement(1)=y_val
630ENDPROC
640
650
660DEF PROCcurve_draw(X1,Y1,X2,Y2,s_factor)
670REM Plot a curve which is a quadrant of the circle.
680command_stack%(1)=curve_draw
690PRINT TAB(0,30) "Curve"
700depth=depth_of_cut
710REM Calculate the radius of the quadrant.
720radius=(SQR((X2-X1)^2+(Y2-Y1)^2))/SQR(2)
730radius%=radius*s_factor
740ON ERROR GOTO 1200
750angle=DEG(ATN((Y2-Y1)/(X2-X1)))
760IF X2<X1 THEN angle=angle+180
770REM Calculate the position of the qudrant centre.
780center_clock_x=X1+radius*COS(RAD(angle-45)):center_clock_y=Y1+radius*SIN(RAD(angle-45)):cen
ter_anticlock_x=X1+radius*COS(RAD(angle+45)):center_anticlock_y=Y1+radius*SIN(RAD(angle+45))
790center_clock_x%=center_clock_x*s_factor:center_clock_y%=center_clock_y*s_factor
800center_anticlock_x%=center_anticlock_x*s_factor:center_anticlock_y%=center_anticlock_y*s_fa
ctor
810VDU5
820REM Print C and A at quadrant centres.
830MOVE center_clock_x%,center_clock_y%
840PRINT "C"
850MOVE center_anticlock_x%,center_anticlock_y%
860PRINT "A"
870VDU4
880CLS
890PRINT TAB(0,2)"Center C":PRINT TAB(0,3)"or A ?";:REPEAT:center$=GET$:PRINT center$:UNTIL ce
nter$="C" OR center$="A"
900PRINT TAB(0,30)"Curve"
910VDU23,1,0;0;0;0;
920GCOL0,0
930VDU5
940MOVE center_clock_x%,center_clock_y%
950PRINT "C"
960MOVE center_anticlock_x%,center_anticlock_y%
970PRINT "A"
980GCOL0,1
990VDU4
1000REM Plot the quadrant in 1 degree increment.
1010IF center$="A" THEN start_angle_pos=angle-135:end_angle_pos=angle-45:angle_inc=1:circle_cen
ter_x%=center_anticlock_x*s_factor:circle_center_y%=center_anticlock_y*s_factor
1020IF center$="C" THEN start_angle_pos=angle+135:end_angle_pos=angle+45:angle_inc=-1:circle_ce
nter_x%=center_clock_x*s_factor:circle_center_y%=center_clock_y*s_factor
1030VDU 29,circle_center_x%;circle_center_y%;
1040MOVE radius%*COS(RAD(start_angle_pos)),radius%*SIN(RAD(start_angle_pos))
1050FOR angle_val=start_angle_pos TO end_angle_pos STEP angle_inc
1060DRAW radius%*COS(RAD(angle_val)),radius%*SIN(RAD(angle_val))
1070NEXT
1080VDU 29,0;0;
1090IF center$="C" THEN g$="G02" ELSE g$="G03"
1100x%=(X2+.001)*100:x$=FNformat(x%,"X"):y%=(Y2+.001)*100:y$=FNformat(y%,"Y"):i%=(ABS(center_cl
ock_x-X1)+.001)*100:i$=FNformat(i%,"I"):j%=(ABS(center_clock_y-Y1)+.001)*100:j$=FNformat(j%,"J")

1110mod_data_block$=n$+"G01"+z$+n$+g$+x$+y$+i$+j$
```

```
1120unmod_data_block$=n$+g$+x$+y$+i$+j$
1130IF command_stack%(2)=move OR command_stack%(2)=depth_change THEN data_block$=mod_data_block
$ ELSE data_block$=unmod_data_block$
1140PRINT£data_file_ch%,data_block$
1150IF data_block$=mod_data_block$ THEN data_block_len%(1)=41 ELSE data_block_len%(1)=30
1160PROCassen_var_update(X2,Y2)
1170VDU23,1,0;0;0;0;
1180CLS
1190GOTO 130
1200IF ERR=18 AND Y2>Y1 THEN angle=90 ELSE angle=-90
1210GOTO 880
1220ENDPROC
1230
1240
1250DEF PROCdesen_var_update
1260REM Update the variables of the privous six commands.
1270LOCAL I
1280I=1
1290REPEAT
1300command_stack%(I)=command_stack%(I+1)
1310data_block_len%(I)=data_block_len%(I+1)
1320privious_xmovement(I)=privious_xmovement(I+1)
1330privious_ymovement(I)=privious_ymovement(I+1)
1340I=I+1
1350UNTIL I=6
1360ENDPROC
1370
1380
1390DEF PROCdepth
1400REM Change the depth of cut.
1410command_stack%(1)=depth_change
1420PRINT TAB(0,20)"Depth"
1430CLS
1440PRINT TAB(0,7)"New depth"
1450INPUT TAB(0,8)"of cut(mm)";depth_of_cut
1460z%=(depth_of_cut+4)*100:z$=FNformat(z%,"Z-"):data_block_len%(1)=0:CLS:PROCtext_screen
1470ENDPROC
1480
1490
1500DEF PROCend
1510REM End the design session.
1520PRINT TAB(0,30)"End":data_block$=n$+"M02"+n$+":":PRINT£data_file_ch%,data_block$:data_block
$="END":PRINT£data_file_ch%,data_block$:CLOSE£data_file_ch%
1530*FX4,0
1540MODE 7
1550@%=10
1560finished=TRUE
1570ENDPROC
1580
1590
1600DEF PROCfeedrate
1610REM Change the existing feedrate.
1620command_stack%(1)=depth_change
1630PRINT TAB(0,20)"Feedrate"
1640CLS
1650PRINT TAB(0,2)"New"
1660INPUT TAB(0,3)"feedrate  ";feedrate%
1670f$="F"+RIGHT$(FNformat(feedrate%,"F"),3)
1680data_block$=n$+f$
1690PRINT£data_file_ch%,data_block$
1700data_block_len%(1)=7
```

```
1710PROCassen_var_update(x_movement,y_movement)
1720CLS
1730PROCtext_screen
1740ENDPROC
1750
1760
1770DEF FNformat(V%,C$)
1780REM Format the data for CNC-1000.
1790IF V%<10 THEN =C$+"0000"+STR$(V%)
1800IF V%>=10 AND V%<100 THEN =C$+"000"+STR$(V%)
1810 IF V%>=100 AND V%<1000 THEN =C$+"00"+STR$(V%)
1820IF V%>=1000 AND V%<10000 THEN =C$+"0"+STR$(V%)
1830IF V%>=10000 AND V%<100000 THEN =C$+STR$(V%)
1840
1850
1860DEF PROCinitialize
1870REM Define various parameters.
1880MODE 4
1890LOCAL row,column,scale_factor_len,scale_factor_wid,grid_spacing
1900REM Define graphic and text windows.
1910VDU23,1,0;0;0;0;:VDU28,30,31,39,0:VDU24,0;0;1100;1023;:VDU19,0,2,0,0,0:VDU19,1,5,0,0,0
1920REM Define colours.
1930COLOUR 0:COLOUR 129:GCOL 0,1:GCOL 0,128:CLG:CLS
1940REM Draw the graphic and text window borders.
1950MOVE 0,0:DRAW 0,1023:DRAW 960,1023:DRAW 960,0:DRAW 0,0
1960VDU29,0;0;
1970REM Calculate the scale factor.
1980scale_factor_len=960/workpiece_len
1990scale_factor_wid=1023/workpiece_wid
2000IF scale_factor_len<scale_factor_wid THEN scale_factor=scale_factor_len ELSE scale_factor=s
cale_factor_wid
2010MOVE scale_factor*workpiece_len,0
2020DRAW scale_factor*workpiece_len,scale_factor*workpiece_wid
2030DRAW 0,scale_factor*workpiece_wid
2040IF scale_factor<12 THEN grid_spacing=10 ELSE grid_spacing=5
2050IF scale_factor<14 THEN unit_movement=0.5 ELSE unit_movement=0.2
2060REM Print the grid on the graphic window.
2070FOR column=0 TO 15*grid_spacing STEP grid_spacing
2080FOR row=0 TO 25*grid_spacing STEP grid_spacing
2090PLOT 69,row*scale_factor,column*scale_factor
2100NEXT row:NEXT column
2110PRINT;grid_spacing;" mm grid"
2120data_file_ch%=OPENOUT(pp_filename$)
2130z%=(depth_of_cut+4)*100:z$=FNformat(z%,"Z-"):f$="F"+RIGHT$(FNformat(feedrate%,"F"),3):first
_data_block$="%"+n$+"N001G00"+f$+"S2T1M03"
2140PRINT£data_file_ch%,first_data_block$
2150CLS
2160ENDPROC
2170
2180
2190DEF PROCline_draw(X,Y,s_factor)
2200REM Plot a straight line.
2210command_stack%(1)=line_draw
2220PRINT TAB(0,30) "Line"
2230depth=depth_of_cut:x%=(X+.001)*100:y%=(Y+.001)*100:x$=FNformat(x%,"X"):y$=FNformat(y%,"Y")
2240mod_data_block$=n$+"G01"+z$+n$+x$+y$
2250unmod_data_block$=n$+"G01"+x$+y$
2260IF command_stack%(2)=move OR command_stack%(2)=depth_change THEN data_block$=mod_data_block
$ ELSE data_block$=unmod_data_block$
2270PRINT£data_file_ch%,data_block$
2280MOVE last_xpos%,last_ypos%
```

```
2290DRAWX*s_factor,Y*s_factor
2300IF data_block$=mod_data_block$ THEN data_block_len%(1)=26 ELSE data_block_len%(1)=18
2310PROCassen_var_update(X,Y)
2320PRINT TAB(0,30)"          "
2330ENDPROC
2340
2350
2360DEF PROCmove(X,Y,s_factor)
2370REM Go to a new position without plotting (i.e. move the cutting head without cutting).
2380command_stack%(1)=move
2390PRINT TAB(0,30) "Move"
2400depth=0:x%=(X+.001)*100:y%=(Y+.001)*100:x$=FNformat(x%,"X"):y$=FNformat(y%,"Y")
2410mod_data_block$=n$+"G00"+"Z00000"+n$+x$+y$
2420unmod_data_block$=n$+x$+y$
2430IF command_stack%(2)=move THEN data_block$=unmod_data_block$ ELSE data_block$=mod_data_bloc
k$
2440PRINT£data_file_ch%,data_block$
2450MOVE last_xpos%,last_ypos%
2460PLOT 70,X*s_factor,Y*s_factor
2470IF data_block$=mod_data_block$ THEN data_block_len%(1)=25 ELSE data_block_len%(1)=15
2480PROCassen_var_update(X,Y)
2490PRINT TAB(0,30)"          "
2500ENDPROC
2510
2520
2530DEF PROCprint
2540REM Provide a hard copy of drawing.
2550PRINT TAB(0,17)"*********"
2560PRINT TAB(0,18)"Job name:"
2570PRINT TAB(0,19);pp_filename$
2580PRINT TAB(0,20)"*********"
2590VDU24,0;0;1279;1023;
2600*SCDUMP
2610PRINT TAB(0,30)"          "
2620VDU24,0;0;1100;1023;
2630PRINT TAB(0,16)"          "
2640PRINT TAB(0,17)"          "
2650PRINT TAB(0,18)"          "
2660PRINT TAB(0,19)"          "
2670PRINT TAB(0,20)"          "
2680PROCend
2690ENDPROC
2700
2710
2720DEF PROCpart_info
2730REM Get the workpiece related data.
2740CLS
2750PRINT TAB(4,2)"Enter workpeice particulars."
2760PRINT TAB(4,3)"----------------------------"
2770REPEAT:PRINT TAB(1,4):INPUT''"Length(mm)              :"workpiece_len:UNTIL workpiece_len>0
AND workpiece_len<=200
2780REPEAT:PRINT TAB(1,7):INPUT"Width (mm)              :"workpiece_wid:UNTIL workpiece_wid>0 AN
D workpiece_wid<=100
2790PRINT TAB(1,8):INPUT"Depth of cut(mm)        :"depth_of_cut
2800PRINT TAB(1,9):INPUT"Feedrate(mm/min)        :"feedrate%
2810REPEAT:PRINT TAB(1,10):INPUT"Part program filename  :"pp_filename$:UNTIL LEN(pp_filename$)<
=5 AND LEN(pp_filename$)>0
2820ENDPROC
2830
2840
2850DEF PROCremove(X2,Y2,X1,Y1,s_factor)
```

```
2860REM Remove an already plotted line or curve.
2870PRINT TAB(0,30) "Remove"
2880PTR£data_file_ch%=PTR£data_file_ch%-data_block_len%(1)
2890REM If previous command was a plot then plot it again in the background colour.
2900IF command_stack%(2)=curve_draw THEN GOTO 2980
2910IF command_stack%(2)=depth_change THEN GOTO 3070
2920GCOL0,0
2930x1%=X1*s_factor:y1%=Y1*s_factor:x2%=X2*s_factor:y2%=Y2*s_factor
2940MOVE x1%,y1%
2950DRAW x2%,y2%
2960GCOL0,1
2970GOTO 3070
2980VDU29,circle_center_x%;circle_center_y%;
2990GCOL0,0
3000MOVE radius%*COS(RAD(end_angle_pos)),radius%*SIN(RAD(end_angle_pos))
3010FOR angle_val=end_angle_pos TO start_angle_pos STEP -angle_inc
3020DRAW radius%*COS(RAD(angle_val)),radius%*SIN(RAD(angle_val))
3030NEXT
3040GCOL0,1
3050VDU29,0;0;
3060VDU23,1,0;0;0;0;
3070PROCdesen_var_update
3080x_movement=X2:y_movement=Y2
3090PRINT TAB(0,30) "          "
3100ENDPROC
3110
3120
3130DEF PROCtext_screen
3140REM Print the menu on the text window.
3150*FX4,1
3160PRINT TAB(0,1)"Move    :";"M"
3170PRINT TAB(0,3)"Line drw:";"L"
3180PRINT TAB(0,5)"Curv drw:";"C"
3190PRINT TAB(0,7)"Remove  :";"R"
3200PRINT TAB(0,9)"Depth   :";"D"
3210PRINT TAB(0,11)"Feedrate:";"F"
3220PRINT TAB(0,13)"End     :";"E"
3230PRINT TAB(0,15)"Print   :";"P"
3240PRINT TAB(0,29)"    mm/min"
3250ENDPROC
>
```

L.

```
   10REM..............................CellTalk...............................
   20REM This is the communication module. It utilses the CAMP for exchange of information betwe
en entities.
   30MODE 7:OSWORD=&FFF1:OSBYTE=&FFF4:*FX 15,0
   40REM Assign a name to each memory location within the CAMP.
   50HIMEM=&7B00:M=&7B00:logi_assoc=M+0:client_st_no=M+1:client_port=M+2:server_resp_mes=M+3:mes
s_stat=M+4:prog_stat=M+5:in_partbuf=M+6:out_partbuf=M+7:production_period=M+8:upload_prep=M+9:pa
rt_trans=M+10:where_part_trans=M+11
   60server_resp_part=M+12:robot_stat=M+13:trolley_stat=M+14:device_stat=M+15:defect_part_stat=M
+16:nof_defect_part=M+17:total_nof_parts=M+18:nof_processed_parts=M+19
   70warehouse%=20:ready=&FF:not_ready=&00:nothing=&09:busy=&FF:free=&00:connect=&FF:disconnect=
&00:processed=&00:processing=&FF:not_recognised=&01:run_prog=&FF:running=&FF:stop=&00:download_s
ucc=&01:transfer_part=&FF:transfer_part_comp=&00
   80empty=&00:completed=&00:idle=&01:stopped=&02:critical_op_in_progress=&03:device_reset=&04:d
evice_inoperable=&05:uncorrectable_error_det=&06:correctable_error_det=&07:diagnostic_running=&0
8
   90defect=&FF:in_progress=&FF:not_started=nothing:started=&FF
  100from_trolley=&01:to_trolley=&02:full=2
  110robot_st%=14:this_cell$="MILLING"
  120op_TYPE=1:key1=op_TYPE:CC_NAME=2:key2=CC_NAME:CC_CON=3:key3=CC_CON:STAT_NO=4:key4=STAT_NO:N
OF_PP=5:key5=NOF_PP:PP_NAME=6:key6=PP_NAME:SEQ_NO=7:key7=SEQ_NO
  130path1=1:path2=2:path3=3:path4=4
  140FOR I=0 TO 30:M?I=nothing:NEXT I
  150?in_partbuf=0:?out_partbuf=0:?nof_defect_part=0:?total_nof_parts=0:?nof_processed_parts=0
  160nof_processed_parts%=0:nof_parts_tobe_manufactured%=0:Part_processing_time%=0:setup_time%=0

  170max_recs=6:rec_fields=11
  180max_peek_length%=5:max_poke_length%=5:current_rec=1
  190REM Reserve a block of memory for the peek and poke operations.
  200DIM cblock%40,peekbuffer% max_peek_length% ,pokebuffer% max_poke_length%
  210DIM field_cont$(rec_fields),index$(max_recs,2),field$(rec_fields),type$(rec_fields),width$(
rec_fields),dp$(rec_fields),command_stack$(8)
  220END
  230
  240
  250DEF PROCpeek(station%,location%,length%)
  260REM Read the content of the CAMP inthe distant cell controller (or master).
  270peek_result%=FNpeek(station%,location%,length%)
  280IF ?prog_stat=stopped THEN PROCterminate_part_prod
  290ENDPROC
  300DEF FNpeek(station%,location%,length%)
  310LOCAL X%,Y%,A%
  320?cblock%=&81:cblock%?1=00:cblock%!2=station%:cblock%!4=peekbuffer%:cblock%!8=peekbuffer%+le
ngth%:cblock%!12=location%:X%=cblock%:Y%=cblock% DIV 256:A%=&10:CALL OSWORD
  330peekresult%=(U% AND &FF00) DIV 256
  340=peekresult%
  350
  360
  370DEF PROCpoke(station%,location%,length%)
  380REM Write to the CAMP of local or distant entity.
  390poke_result%=FNpoke(station%,location%,length%)
  400IF poke_result%=0 THEN ENDPROC
  410ON poke_result%-&3F GOTO 420,430,440,450,460
  420PRINT''" ** LINE JAMMED, TRYING AGAIN **":SOUND 1,-15,5,15:GOTO 390
  430PRINT''" ** NETWORK ERROR, TRYING AGAIN **":SOUND 1,-15,5,15:GOTO 390
  440PRINT''" ** NOT LISTENING, TRYING AGAIN **":SOUND 1,-15,5,15:GOTO 390
  450PRINT''" ** NO CLOCK, TRYING AGAIN **":SOUND 1,-15,5,15:GOTO 390
  460PRINT''" ** BAD CONTROL BLOCK, TRYING AGAIN **":SOUND 1,-15,5,15:GOTO 390
  470DEF FNpoke(station%,location%,length%)
  480LOCAL X%,Y%,A%
  490?cblock%=&82:cblock%?1=00:cblock%!2=station%:cblock%!4=pokebuffer%:cblock%!8=pokebuffer%+le
ngth%:cblock%!12=location%:X%=cblock%:Y%=cblock% DIV 256:A%=&10:CALL OSWORD
  500PROCdelay(200)
  510REPEAT:A%=&32:U%=USR OSBYTE:UNTIL (U% AND &8000)=0
  520pokeresult%=(U% AND &FF00) DIV 256
  530=pokeresult%
  540
  550
  560DEF PROC_keyboard_poke(station%,mail$)
  570REM Insert a string of characters (a message) to the keyboard buffer of distant entity.
  580?cblock%=&01:cblock%!1=station%:$(cblock%+3)=mail$:X%=cblock%:Y%=cblock% DIV 256:A%=&14:CAL
L OSWORD
  590ENDPROC
  600
  610
  620DEF PROCget_mail
  630REM Receive a mail from the distant entity.
  640*FX 21,0
  650REPEAT:message$="":PRINT".";
  660REPEAT
  670IF ?prog_stat=stopped THEN PROCterminate_part_prod
  680REM Check for the start of data code.
  690key$=INKEY$(0):UNTIL key$="{" OR ?logi_assoc=disconnect
  700IF ?logi_assoc=disconnect THEN 840
  710REPEAT:key$=INKEY$(0)
  720IF ?prog_stat=stopped THEN PROCterminate_part_prod
  730message$=message$+key$
  740REM Check for the end of data code.
  750UNTIL key$="}"
  760REM Process the message.
  770?mess_stat=processing
  780PROCdelay(100)
  790length=LEN(message$)
  800IF LEFT$(message$,1)="{" THEN message$=MID$(message$,2,length-2) ELSE message$=LEFT$(messag
e$,length-1)
  810PRINT"The message received from st. '";?client_st_no;"' is :";message$
  820PROCmail_test(message$)
  830REM End the logical association session.
  840UNTIL ?logi_assoc=disconnect
  850ENDPROC
  860
  870
  880DEF PROCmail_test(mess$)
  890REM Check the meaning of the received message and act upon it accordingly.
  900LOCAL len
  910len=LEN(mess$)
  920IF LEFT$(mess$,4)="SEND" THEN ?server_resp_mes=not_ready:?mess_stat=processed:PROCrespond:E
NDPROC
  930IF mess$="RESET" OR mess$="reset" THEN PROCreset:?mess_stat=processed:ENDPROC
  940IF LEFT$(mess$,9)="DOWN LOAD" OR LEFT$(mess$,9)="down load" THEN IF MID$(mess$,11)<>"" THEN
 file$=MID$(mess$,11):PROCdownload(file$):ENDPROC
  950IF mess$="START" OR mess$="start" THEN PROCstart_milling:?mess_stat=processed:ENDPROC
  960IF mess$="MONITOR" OR mess$="monitor" THEN ?mess_stat=processed:CHAIN"MONITOR":ENDPROC
  970IF LEFT$(mess$,8)="JOB NAME" THEN job_name$=MID$(mess$,10):?mess_stat=processed:PRINT job_n
ame$:ENDPROC
  980IF LEFT$(mess$,9)="NOF PARTS" THEN nof_parts_tobe_manufactured%=VAL(MID$(mess$,10)):?mess_s
tat=processed:ok=TRUE:PRINT nof_parts_tobe_manufactured%:ENDPROC
  990IF LEFT$(mess$,1)="D" THEN PRINT'"_____The requested part dimention is : ";MID$(mess$,2):?m
ess_stat=processed:ENDPROC
 1000IF LEFT$(mess$,1)="O" THEN PRINT'"_____The requested part orentation is : ";MID$(mess$,2):?
mess_stat=processed:ENDPROC
```

```
 1010?mess_stat=not_recognised:PRINT"This message is not defined in the VMD!."
 1020ENDPROC
 1030
 1040
 1050DEF PROCtx_message(local_st%,s%,message$)
 1060REM Send a mail to a distant entity.
 1070REPEAT
 1080REM Establish a logical association first.
 1090loc%=logi_assoc:len%=1:PROCpeek(s%,loc%,len%):UNTIL peekbuffer%?0<>connect
 1100PRINT'"Logical Association between st. '";local_st%;"'";" and st. '";s%;"'"
 1110?pokebuffer%=connect
 1120pokebuffer%?1=local_st%:loc%=logi_assoc:len%=2:PROCpoke(s%,loc%,len%):loc%=server_resp_mes:
len%=1:PRINT'"Waiting for response from st. '";s%;"'...":PROCdelay(10)
 1130REPEAT:PROCpeek(s%,loc%,len%):UNTIL peekbuffer%?0=ready
 1140PRINT"Station '";s%;"' is responded now"
 1150PRINT"The message sent to st. '";s%;"' is (";message$;")"
 1160REM Encode the message before dispatching it.
 1170message$="{"+"{"+message$+"}":PROC_keyboard_poke(s%,message$):PRINT"Waiting for the message
 to be processed in the st. '";s%;"'...":loc%=mess_stat:len%=1:REPEAT:PROCpeek(s%,loc%,len%)
 1180UNTIL peekbuffer%?0=processing
 1190REM Check the distant entity until the message is received and recognised successfully.
 1200REPEAT:PROCpeek(s%,loc%,len%):UNTIL peekbuffer%?0=processed OR peekbuffer%?0=not_recognised
:PRINT"The message at st. '";s%;"' is processed now"
 1210IF peekbuffer%?0=not_recognised THEN PRINT"Station ";s%;" does not recognise this message,
try again."
 1220REM End the session.
 1230loc%=logi_assoc:len%=1:pokebuffer%?0=disconnect:PROCpoke(s%,loc%,len%)
 1240ENDPROC
 1250
 1260
 1270DEF PROCrespond
 1280REM This procedure is used by the cell controllers to respond to a request invoked by dista
nt entity.
 1290client_st%=?client_st_no
 1300IF MID$(mess$,6)="PART ORIENTATION" THEN PROCget_part_orientation:PROCtx_message(this_st%,c
lient_st%,mes$):ENDPROC
 1310IF MID$(mess$,6)="PART DIMENTION" THEN PROCget_part_dimention:PROCtx_message(this_st%,clien
t_st%,mes$):ENDPROC
 1320PRINT"ROUTINE IS NOT DEFINED !."
 1330ENDPROC
 1340
 1350
 1360DEF PROCinfo_request(query$,server_st%)
 1370REM this routine is used by entities which require information from other entities
 1380server=FALSE
 1390PROCtx_message(this_st%,server_st%,query$)
 1400REPEAT
 1410IF ?logi_assoc=connect THEN IF ?client_st_no=server_st% THEN server=TRUE:?server_resp_mes=r
eady:PROCget_mail:?server_resp_mes=not_ready
 1420IF ?logi_assoc=connect THEN IF ?client_st_no<>server_st% THEN ?server_resp_mes=ready:PROCge
t_mail:?server_resp_mes=not_ready
 1430IF ?prog_stat=stopped THEN PROCterminate_part_prod
 1440UNTIL server
 1450ENDPROC
 1460
 1470
 1480DEF PROCget_part_orientation
 1490REM This is used by cell controllers to obtain the orientation of the part which is current
ly being processed in this cell. This information is then provided to the distant entity.
 1500lower=10:upper=99:X$=STR$(FNrnd_num(lower,upper)):lower=10:upper=99:Y$=STR$(FNrnd_num(lower
,upper)):lower=10:upper=99:deg$=STR$(FNrnd_num(lower,upper)):lower=10:upper=99:min$=STR$(FNrnd_n
um(lower,upper))
 1510mes$="O"+"XOO"+X$+"YOO"+Y$+"ANG"+deg$+"."+min$
 1520ENDPROC
 1530
 1540
 1550DEF PROCget_part_dimention
 1560REM This is used by the cell controllers to obtain the dimension of a part currently being
processed for the remote entity.
 1570lower=10:upper=99:L$=STR$(FNrnd_num(lower,upper)):lower=10:upper=99:W$=STR$(FNrnd_num(lower
,upper)):lower=10:upper=99:H$=STR$(FNrnd_num(lower,upper))
 1580mes$="D"+"LOO"+L$+"WOO"+W$+"HOO"+H$
 1590ENDPROC
>
```

L.

```
  10REM..........................MAN_DBASE................................
  20REM This is the Manufacturing Database program.
  30REM Set the maximum number of record to be allocated. One record per cell is used.
  40CLS:max_recs=5
  50REM Set the number of fields per record to be used.
  60rec_fields=18
  70DIM field_cont$(rec_fields),index$(max_recs,2),field$(rec_fields),type$(rec_fields),width$(
rec_fields),dp$(rec_fields),command_stack$(8)
  80REM Set the record pointer to the first record.
  90current_rec=1
 100quit=FALSE
 110CLS:PROCfirst_menu
 120REPEAT
 130REPEAT
 140REM Print the prompt.
 150PRINT'"$";
 160REM Get a command at the $ prompt.
 170INPUT"" command$:UNTIL command$<>""
 180FOR I=1 TO 6:command_stack$(I)="":NEXT I
 190K=0
 200REPEAT
 210s=INSTR(command$," ")
 220K=K+1
 230command_stack$(K)=LEFT$(command$,s-1)
 240command$=MID$(command$,s+1)
 250UNTIL s=0
 260REM Test the received command before calling the appropriate procedure.
 270IF command_stack$(1)="ADD" OR command_stack$(1)="A" THEN PROCadd_rec
 280IF command_stack$(1)="CLOSE" AND command_stack$(3)="" THEN PROCclose_file
 290IF command_stack$(1)="CLOSE" AND command_stack$(3)<>"" THEN PROCclose_file_update_index
 300IF command_stack$(1)="COMPRESS" OR command_stack$(1)="CO" THEN PROCcompress_recs
 310IF command_stack$(1)="CREATE" OR command_stack$(1)="CR" THEN PROCcreate_file
 320IF command_stack$(1)="DELETE" OR command_stack$(1)="DEL"  THEN PROCdelete_rec
 330IF command_stack$(1)="DISPLAY" OR command_stack$(1)="DIS" THEN PROCdisplay_rec
 340IF command_stack$(1)="EDIT" OR command_stack$(1)="E" THEN PROCedit_rec
 350IF command_stack$(1)="FIND" OR command_stack$(1)="F" THEN PROCfind_rec
 360IF command_stack$(1)="GOTO" OR command_stack$(1)="G" THEN PROCgoto_rec
 370IF command_stack$(1)="LIST" OR command_stack$(1)="L" THEN PROClist_recs
 380IF command_stack$(1)="MENU" OR command_stack$(1)="M" THEN PROCsecond_menu
 390IF command_stack$(1)="OPEN" AND command_stack$(2)<>"UPDATE" THEN PROCopen_file
 400IF command_stack$(1)="OPEN" AND command_stack$(2)="UPDATE" THEN PROCopen_update_file
 410IF command_stack$(1)="ORDER" ORcommand_stack$(1)="OR" THEN PROCorder_recs
 420IF command_stack$(1)="QUIT" OR command_stack$(1)="Q" THEN quit=TRUE:GOTO 450
 430IF command_stack$(1)="RETRIEVE" OR command_stack$(1)="R" THEN PROCretrieve_rec
 440IF command_stack$(1)="SHOW" OR command_stack$(1)="S" THEN PROCshow_field
 450UNTIL quit
 460END
 470
 480
 490
 500DEF PROCadd_rec
 510REM Add record to an existing data file.
 520REPEAT
 530PROCadd
 540INPUT TAB(8,23) "Add more (Y/N)" more$
 550UNTIL LEFT$(more$,1)<>"Y" AND LEFT$(more$,1)<>"y"
 560ENDPROC
 570
 580DEF PROCadd
 590CLS
```

```
600limit=8
610FOR I=1 TO n
620PRINT TAB(0,I) field$(I);TAB(limit,I)":"+STRING$(VAL(width$(I))," ")+":"
630NEXT I
640FOR I=1 TO n
650PRINT TAB(limit+1,I);
660IF type$(I)="N" THEN PROCget_num ELSE PROCget_chars
670NEXT I
680PTR£db_ch=point_eof
690rec$=""
700FOR I=1 TO n
710rec$=rec$+field_cont$(I)
720NEXT I
730rec$=" "+rec$
740PRINT£db_ch,rec$
750point_eof=PTR£db_ch
760nofrec$=STR$(VAL(nofrec$)+1)
770nofrec$=FNmodify(nofrec$,4)
780PTR£db_ch=point_nofrec
790PRINT£db_ch,nofrec$
800IF index_flag=FALSE THEN ENDPROC
810
820REM Sort the insertions into index array.
830I=0
840REPEAT
850I=I+1
860UNTIL index$(I,1)>field_cont$(VAL(keyfield$)) OR I=VAL(nofrec$)
870REM Move up all index records.
880FOR J=VAL(nofrec$) TO I STEP -1
890index$(J+1,1)=index$(J,1):index$(J+1,2)=index$(J,2)
900NEXT J
910index$(I,1)=field_cont$(VAL(keyfield$)):index$(I,2)=nofrec$
920ENDPROC
930
940
950DEF PROCchop_arraytable
960REM Perform a logarithmic search(i.e. x=log n/log 2).
970mid=(first%+last%) DIV 2
980IF index$(mid,1)=search_spec$ THEN stop_searching=TRUE:found=TRUE:ENDPROC
990IF index$(mid,1)>search_spec$ THEN last%=mid-1 ELSE first%=mid+1
1000IF first%>last% THEN stop_searching=TRUE
1010ENDPROC
1020
1030
1040DEF PROCclose_file
1050REM Close a file without updating it.
1060IF command_stack$(2)="" THEN PRINT"SYNTAX ERROR" ELSE PROCclose(command_stack$(2))
1070ENDPROC
1080
1090DEF PROCclose(file$)
1100CLOSE£db_ch
1110CLS
1120ENDPROC
1130
1140
1150DEF PROCclose_file_update_index
1160REM Close a file it has been updated.
1170IF command_stack$(2)="" OR command_stack$(3)<>"UPDATE" OR command_stack$(4)="" OR command_s
tack$(5)="" THEN PRINT"SYNTAX ERROR" ELSE PROCclose_update(command_stack$(2),command_stack$(4),c
ommand_stack$(5))
1180ENDPROC
```

```
1190
1200DEF PROCclose_update(file$,mod_idx$,keyfield$)
1210CLS
1220PRINT TAB(1,23) "Wait..."
1230PROCorder(mod_idx$,keyfield$)
1240CLOSE£db_ch
1250CLS
1260ENDPROC
1270
1280
1290DEF PROCcompress_recs
1300REM Overwrite on records being marked for deletion.
1310com$=command_stack$(2)
1320IF com$="" THEN PRINT "SYNTAX ERROR " ELSE PROCcompress(com$)
1330ENDPROC
1340
1350DEF PROCcompress(file$)
1360LOCAL rec_del,oldpoint,newpoint,rdel
1370PRINT TAB(0,23) STRING$(30," "):PRINT TAB(1,23) "Wait..."
1380oldpoint=1:newpoint=1
1390rec_del=0
1400FOR I=1 TO VAL(nofrec$)
1410REM Read record
1420PTR£db_ch=point_nofrec+6+(oldpoint-1)*(rec_len+2)
1430INPUT£db_ch,record$
1440oldpoint=oldpoint+1
1450REM If not marked for deletion then write it.
1460IF LEFT$(record$,1)="*" THEN recs_del=recs_del+1
1470IF LEFT$(record$,1)=" " THEN PTR£db_ch=point_nofrec+6+(newpoint-1)*(rec_len+2):PRINT£db_ch,
record$:newpoint=newpoint+1
1480NEXT I
1490PTR£db_ch=point_nofrec
1500nofrec$=STR$(VAL(nofrec$)-recs_del)
1510nofrec$=FNmodify(nofrec$,4)
1520PRINT£db_ch,nofrec$
1530point_eof=VAL(nofrec$)*(rec_len+2)+point_nofrec+6
1540IF index_flag=FALSE THEN ENDPROC
1550REM Tidy up the index array.
1560rdel=0:J=1
1570FOR I=1 TO VAL(nofrec$)+recs_del
1580IF LEFT$(index$(I,2),1)<>"*" THEN index$(J,1)=index$(I,1):index$(J,2)=index$(I,2):J=J+1
1590IF LEFT$(index$(I,2),1)="*" THEN rdel=rdel+1
1600NEXT I
1610IF recs_del<>rdel THEN PRINT"The index is being corrupted..reindexing":PROCindex(index_file
$,keyfield$)
1620CLS
1630ENDPROC
1640
1650
1660DEF PROCcreate_new_job
1670REM Create a new job.
1680IF command_stack$(2)<>"" THEN db_file$=command_stack$(2) ELSE INPUT "ENTER FILE NAME..."db_
file$
1690PRINT TAB(0,23) STRING$(20," "):PRINT TAB(1,23) "Wait..."
1700PROCcreate(db_file$)
1710mod_idx$=db_file$+"_IND"
1720keyfield$="1"
1730index_ch=OPENOUT(mod_idx$)
1740PRINT "Enter record now "
1750CLOSE£db_ch
1760PROCopen(db_file$):PROCadd_rec
```

```
1770PRINT TAB(0,23) STRING$(30," "):PRINT TAB(1,23) "Wait..."
1780PRINT£index_ch,keyfield$
1790FOR I=1 TO VAL(nofrec$)
1800PRINT£index_ch,index$(I,1)
1810PRINT£index_ch,index$(I,2)
1820NEXT I
1830CLOSE£index_ch
1840mod_idx$=db_file$+"_INS"
1850keyfield$="7"
1860PROCorder(mod_idx$,keyfield$)
1870CLOSE£db_ch
1880PROCsecond_menu
1890ENDPROC
1900
1910
1920DEF PROCcreate_file
1930REM Create a datafile.
1940IF command_stack$(2)<>"" THEN db_file$=command_stack$(2) ELSE INPUT "ENTER FILE NAME..."db_
file$
1950PRINT TAB(0,23) STRING$(20," "):PRINT TAB(1,23) "Wait..."
1960PROCcreate(db_file$)
1970CLS
1980REPEAT
1990PRINT TAB(4,10)"Enter index file name ";
2000INPUT mod_idx$
2010UNTIL LEN(mod_idx$)<=10
2020REPEAT
2030INPUT TAB(4,11)"Enter index field number";keyfield$
2040UNTIL keyfield$<>""
2050PRINT TAB(0,23) STRING$(20," "):PRINT TAB(1,23) "Wait..."
2060index_ch=OPENOUT(mod_idx$)
2070CLS
2080PRINT TAB(4,10)"Enter record now"
2090CLOSE£db_ch
2100PROCopen(db_file$):PROCadd_rec
2110PRINT£index_ch,keyfield$
2120FOR I=1 TO VAL(nofrec$)
2130PRINT£index_ch,index$(I,1)
2140PRINT£index_ch,index$(I,2)
2150NEXT I
2160CLOSE£index_ch
2170ENDPROC
2180
2190DEF PROCcreate(file$)
2200PRINT TAB(0,23) STRING$(20," "):PRINT TAB(1,23) "Wait..."
2210LOCAL name$,width$,dp$,type$,J
2220IF LEN(file$)>10 THEN PRINT "FATAL ERROR _ FILE NAME TOO LONG":STOP
2230rec_len=0
2240db_ch=OPENOUT(file$)
2250IF db_ch=0 THEN PRINT"FILE CANNOT BE OPENED":ENDPROC
2260CLS
2270REPEAT
2280good=TRUE
2290REPEAT
2300INPUT TAB(4,10)"Enter number of fields "n$
2310UNTIL n$<>""
2320FOR J=1 TO LEN(n$)
2330IF INSTR("0123456789",MID$(n$,J,1))=0 THEN good=FALSE
2340IF VAL(n$)>32 THEN good=FALSE
2350NEXT J
2360PRINT TAB(4,10) STRING$(23+LEN(n$)," ")
```

```
2370UNTIL good
2380n$=FNmodify(n$,2)
2390PRINT£db_ch,n$
2400CLS
2410PRINT'''"NAME    TYPE    WIDTH    DPLACES"
2420FOR J=1 TO VAL(n$)
2430INPUT TAB(0,J+4)""name$:name$=FNmodify(name$,8)
2440REPEAT
2450REM Get information about the header record.
2460INPUT TAB(10,J+4)""type$
2470type$=FNmodify(type$,1)
2480UNTIL type$="C" OR type$="N"
2490REPEAT
2500INPUT TAB(18,J+4)""width$
2510width$=FNmodify(width$,2)
2520rec_len= rec_len+VAL(width$)
2530UNTIL INSTR("0123456789",LEFT$(width$,1))>0 AND INSTR("0123456789",RIGHT$(width$,1))>0
2540REPEAT
2550INPUT TAB(26,J+4)""dp$
2560IF dp$<>""THEN dp$=LEFT$(dp$,1) ELSE dp$=" "
2570UNTIL INSTR(" 0123456789",dp$)>0
2580PRINT TAB(1,J+4) STRING$(40," ")
2590PRINT TAB(0,J+4) name$;TAB(10,J+4) type$;TAB(18,J+4) width$;TAB(26,J+4) dp$
2600PRINT£db_ch,name$+type$+width$+dp$
2610NEXT J
2620rec_len=rec_len+1
2630nofrec$="0000":PRINT£db_ch,nofrec$
2640dummy$=STRING$(rec_len," ")
2650FOR R=1 TO max_recs:PRINT£db_ch,dummy$:NEXT R
2660PRINT TAB(1,23) "Wait..."
2670index_flag=TRUE
2680ENDPROC
2690
2700
2710DEF FNmodify(s$,n)
2720REM Truncate the string length.
2730IF LEN(s$)<n THEN s$=s$+STRING$(n-LEN(s$)," ")
2740IF LEN(s$)>n THEN s$=LEFT$(s$,n)
2750=s$
2760
2770
2780DEF PROCdelete_rec
2790REM Mark the records for deletion.
2800LOCAL s
2810IF command_stack$(2)="" THEN PROCdelete(current_rec)
2820s=INSTR(command_stack$(2),";")
2830IF s=0 THEN PROCdelete(VAL(command_stack$(2))):ENDPROC
2840del_s=VAL(LEFT$(command_stack$(2),s-1))
2850del_f=VAL(MID$(command_stack$(2),s+1))
2860FOR I=del_s TO del_f
2870PROCdelete(I)
2880NEXT I
2890ENDPROC
2900
2910DEF PROCdelete(rec)
2920PTR£db_ch=point_nofrec+6+(rec-1)*(rec_len+2)
2930INPUT£db_ch,record$:PRINT record$
2940REM Add a * in front of records to be deleted.
2950record$="*"+MID$(record$,2)
2960PTR£db_ch=point_nofrec+6+(rec-1)*(rec_len+2)
2970PRINT£db_ch,record$
```

```
2980IF index_flag=FALSE THEN ENDPROC
2990REM remove entry from index file
3000J=0
3010REPEAT J=J+1
3020IF LEFT$(index$(J,2),1)="*" THEN r=VAL(MID$(index$(J,2),2)) ELSE r=VAL(index$(J,2))
3030UNTIL r=rec
3040index$(J,2)="*"+index$(J,2)
3050ENDPROC
3060
3070
3080DEF PROCdisplay_rec
3090REM Display a record.
3100com$=command_stack$(2)
3110IF com$="" THEN PROCdisplay(current_rec) ELSE IF VAL(com$)=0 THEN PRINT "SYNTAX ERROR" ELSE
PROCdisplay(VAL(com$))
3120ENDPROC
3130DEF PROCdisplay(rec)
3140PTR£db_ch=point_nofrec+6+(rec-1)*(rec_len+2)
3150INPUT£db_ch,record$
3160PRINT FNmodify(STR$(rec),3);":"+record$
3170ENDPROC
3180
3190
3200DEF PROCedit_rec
3210REM Edit the contents of a record.
3220com$=command_stack$(2)
3230IF com$="" THEN PRINT "SYNTAX ERROR"ELSE PROCedit(VAL(com$))
3240ENDPROC
3250
3260DEF PROCedit(rec)
3270LOCAL limit,far
3280PTR£db_ch=point_nofrec+6+(rec-1)*(rec_len+2)
3290INPUT£db_ch,record$
3300record$=MID$(record$,2)
3310CLS
3320limit=8
3330FOR I=1 TO n
3340PRINT TAB(0,I) field$(I);TAB(limit,I)":"+STRING$(VAL(width$(I))," ")+":"
3350NEXT I
3360far=1
3370FOR I=1 TO n
3380field_cont$(I)=MID$(record$,far,VAL(width$(I)))
3390far=far+VAL(width$(I))
3400PRINT TAB(limit+1,I);field_cont$(I);
3410PRINT TAB(2,20);"press C to change field ";I;" Return to leave alone";
3420REPEAT
3430g$=GET$
3440UNTIL g$="c" OR g$="C" OR ASC(g$)=13
3450PRINT TAB(limit+1,I);
3460 IF ASC(g$)=13 THEN PRINT TAB(2,20);"No change "+STRING$(50," ");ELSE IF type$(I)="N" THEN
PROCget_num ELSE PROCget_chars
3470NEXT I
3480rec$=""
3490FOR I=1 TO n
3500rec$=rec$+field_cont$(I)
3510NEXT I
3520rec$=" "+rec$
3530PTR£db_ch=point_nofrec+6+(rec-1)*(rec_len+2)
3540PRINT£db_ch,rec$:CLS
3550ENDPROC
3560
```

```
3570
3580DEF PROCfind_rec
3590REM Search for a record.
3600com$=command_stack$(2)
3610IF com$="" THEN PRINT "SYNTAX ERROR" ELSE PROCfind(com$)
3620ENDPROC
3630
3640DEF PROCfind(search_spec$)
3650LOCAL first%,last%,d,stop_searching,found
3660IF LEN(search_spec$)<VAL(width$(key%)) THEN d=VAL(width$(key%))-LEN(search_spec$):FOR I=1 T
O d:search_spec$=search_spec$+" ":NEXT I
3670first%=1
3680last%=VAL(nofrec$)
3690stop_searching=FALSE:found=FALSE
3700REPEAT
3710PROCchop_arraytable
3720UNTIL stop_searching
3730IF found THEN current_rec=VAL(index$(mid,2)):PROCdisplay(current_rec) ELSE PRINT"RECORD NOT
FOUND"
3740ENDPROC
3750
3760
3770DEF FNget_field(rec,field)
3780REM Find field of a record.
3790PTR£db_ch=point_nofrec+6+(rec-1)*(rec_len+2)
3800INPUT£db_ch,record$
3810record$=MID$(record$,2)
3820begin=1:IF field>1 THEN FOR I=1 TO field-1:begin=begin+VAL(width$(I)):NEXT I
3830field_cont$(field)=MID$(record$,begin,VAL(width$(field)))
3840=field_cont$(field)
3850
3860
3870DEF PROCgoto_rec
3880REM Change the positon of the record pointer.
3890v=VAL(command_stack$(2))
3900IF v=0 THEN PROCgoto(1):ENDPROC
3910PROCgoto(v)
3920ENDPROC
3930
3940DEF PROCgoto(num)
3950current_rec=num
3960ENDPROC
3970
3980
3990DEF PROClist_recs
4000REM Give a list of records on the datafile.
4010rec=0
4020line=0
4030REPEAT
4040line=line+1:rec=rec+1
4050IF VAL(index$(rec,2))<>0 THEN PROCdisplay(VAL(index$(rec,2)))
4060IF line=20 THEN PRINT''"Press key to continue":g=GET:line=0
4070UNTIL rec=VAL(nofrec$)
4080ENDPROC
4090
4100
4110DEF PROCfirst_menu
4120CLS
4130PRINT TAB(0,5)"Do you want to define a new job (Y/N)"
4140REPEAT:INPUT g$:UNTIL g$="Y" OR g$="y" OR g$="N" OR g$="n"
4150IF g$="Y" OR g$="y" THEN CLS:PRINT TAB(5,11):INPUT "Enter the new job name (e.g. JOB1)";com
```

```
mand_stack$(2):command_stack$(1)="CREATE":PROCcreate_new_job ELSE PROCsecond_menu
 4160ENDPROC
 4170
 4180
 4190DEF PROCsecond_menu
 4200REM A menu of commands is allowed to be entered.
 4210CLS
 4220PRINT TAB(0,0)"Choose command/s and enter in the following format.       (Page1)"
 4230PRINT"----------Press M for Menu--------------"
 4240PRINT'"_  <ADD> "
 4250PRINT"_  <CLOSE> <filename>"
 4260PRINT"_  <CLOSE> <filename> <UPDATE> <index filename> <field num>"
 4270PRINT"_  <COMPRESS> <filename>"
 4280PRINT"_  <CREATE> <filename> "
 4290PRINT"_  <DELETE> <rec num> or <DELETE> <from rec> <;> <to rec>"
 4300PRINT"_  <DISPLAY> <rec num>"
 4310PRINT"_  <EDIT> <rec num>"
 4320PRINT"_  <FIND> <field content>"
 4330PRINT"_  <GOTO> <rec num>"
 4340PRINT"_  <LIST> or <LIST> <FOR> <condition>"
 4350PRINT"_  <OPEN> <filename> <INDEX> <index filename>"
 4360PRINT"_  <OPEN> <UPDATE> <filename> <INDEX> <index filename>"
 4370PRINT'"......Press P to go to next page......"
 4380G$=GET$:IF G$="P" OR G$="p" THEN PROCsecond_menu_p2
 4390ENDPROC
 4400
 4410DEF PROCsecond_menu_p2
 4420CLS:PRINT TAB(0,0)"Choose command/s and enter in the following format.       (Page2)"
 4430PRINT"----------Press M for Menu--------------"
 4440PRINT'"_ '<ORDER> <ON> <field num> <TO> <index filename>"
 4450PRINT"_  <QUIT>"
 4460PRINT"_  <RETRIEVE> <rec num>"
 4470PRINT"_  <SHOW> <rec num> <field num>"
 4480ENDPROC
 4490
 4500
 4510DEF PROCopen_file
 4520REM Open a file for reading purposes only.
 4530IF command_stack$(3)="INDEX" THEN PROCopen(command_stack$(2)):PROCreadindex(command_stack$(
4)) ELSE PRINT "SYNTAX ERROR"
 4540ENDPROC
 4550
 4560
 4570DEF PROCopen(file$)
 4580PRINT TAB(0,23) STRING$(30," "):PRINT TAB(1,23) "Wait..."
 4590LOCAL I,rec$
 4600db_ch=OPENIN(file$)
 4610INPUT£db_ch,n$
 4620n=VAL(n$)
 4630rec_len=0
 4640FOR I=1 TO n
 4650INPUT£db_ch,rec$
 4660field$(I)=LEFT$(rec$,8)
 4670type$(I)=MID$(rec$,9,1)
 4680width$(I)=MID$(rec$,10,2)
 4690rec_len=rec_len+VAL(width$(I))
 4700dp$(I)=MID$(rec$,12)
 4710NEXT I
 4720rec_len=rec_len+1
 4730point_nofrec=PTR£db_ch
 4740INPUT£db_ch,nofrec$
```

```
4750point_eof=VAL(nofrec$)*(rec_len+2)+point_nofrec+6
4760ENDPROC
4770
4780
4790DEF PROCopen_update_file
4800REM Open a datafile for reading and writing.
4810IF command_stack$(2)="UPDATE" THEN PROCopen_update(command_stack$(3)):PROCreadindex(command
_stack$(5)) ELSE PRINT "SYNTAX ERROR"
4820ENDPROC
4830
4840DEF PROCopen_update(file$)
4850PRINT TAB(0,23) STRING$(30," "):PRINT TAB(1,23) "Wait..."
4860LOCAL I,rec$
4870db_ch=OPENUP(file$)
4880INPUT£db_ch,n$
4890n=VAL(n$)
4900rec_len=0
4910FOR I=1 TO n
4920INPUT£db_ch,rec$
4930field$(I)=LEFT$(rec$,8)
4940type$(I)=MID$(rec$,9,1)
4950width$(I)=MID$(rec$,10,2)
4960rec_len=rec_len+VAL(width$(I))
4970dp$(I)=MID$(rec$,12)
4980NEXT I
4990rec_len=rec_len+1
5000point_nofrec=PTR£db_ch
5010INPUT£db_ch,nofrec$
5020point_eof=VAL(nofrec$)*(rec_len+2)+point_nofrec+6
5030ENDPROC
5040
5050
5060DEF PROCorder_recs
5070REM Arrange the order in which the records will be accessed or displayed.
5080IF command_stack$(2)="" THEN PROCorder(index_file$,keyfield$):ENDPROC
5090IF command_stack$(2)<>"ON" AND command_stack$(4)<>"TO" THEN PRINT"SYNTAX ERROR":ENDPROC
5100PROCorder(command_stack$(5),command_stack$(3))
5110ENDPROC
5120
5130DEF PROCorder(index_file$,keyfield$)
5140LOCAL begin,length
5150PRINT TAB(0,23) STRING$(30," "):PRINT TAB(1,23) "Wait..."
5160index_ch=OPENOUT(index_file$)
5170PRINT£index_ch,keyfield$
5180key%=VAL(keyfield$)
5190PTR£db_ch=point_nofrec+6:REM point to start of data
5200begin=1
5210FOR K=1 TO VAL(keyfield$)-1
5220begin=begin+VAL(width$(K))
5230NEXT K
5240IF VAL(keyfield$)=1 THEN begin=1
5250length=VAL(width$(VAL(keyfield$)))
5260FOR I=1 TO VAL(nofrec$)
5270INPUT£db_ch,rec$
5280rec$=MID$(rec$,2):REM peel off deletion flag
5290index$(I,1)=MID$(rec$,begin,length):index$(I,2)=STR$(I)
5300NEXT I
5310index_flag=TRUE
5320PROCsort
5330ENDPROC
5340REM
```

```
5350
5360
5370DEF PROCretrieve_rec
5380REM Retrieve a record which is being marked for deletion.
5390com$=command_stack$(2)
5400IF com$="" THEN PRINT "SYNTAX ERROR "ELSE PROCretrieve(VAL(com$))
5410ENDPROC
5420
5430DEF PROCretrieve(rec)
5440PTR£db_ch=point_nofrec+6+(rec-1)*(rec_len+2)
5450INPUT£db_ch,record$
5460record$=" "+MID$(record$,2)
5470PTR£db_ch=point_nofrec+6+(rec-1)*(rec_len+2)
5480PRINT£db_ch,record$
5490IF index_flag=FALSE THEN ENDPROC
5500REM Replace entry to index file
5510J=0
5520REPEAT J=J+1
5530IF LEFT$(index$(J,2),1)="*" THEN v=VAL(MID$(index$(J,2),2)) ELSE v=VAL(index$(J,2))
5540UNTIL v=rec
5550index$(J,2)=MID$(index$(J,2),2)
5560ENDPROC
5570
5580
5590DEF PROCreadindex(index_file$)
5600REM Fill the contents of the index array with data.
5610PRINT TAB(1,23) "Wait..."
5620 LOCAL index_ch,I
5630IF index_file$=""THEN index_flag=FALSE:ENDPROC ELSE index_flag=TRUE
5640index_ch=OPENIN(index_file$)
5650INPUT£index_ch,keyfield$
5660key%=VAL(keyfield$)
5670IF keyfield$=""THEN PRINT"BAD FORMED INDEX":STOP
5680FOR I=1 TO VAL(nofrec$)
5690INPUT£index_ch,index$(I,1)
5700INPUT£index_ch,index$(I,2)
5710NEXT I
5720CLOSE£index_ch
5730CLS
5740ENDPROC
5750
5760
5770DEF PROCget_num
5780REM Get a numeric value for the current field of selected record.
5790LOCAL J,G$,st$,hor,ver
5800after=VAL(dp$(I))
5810before=VAL(width$(I))-after:REM Implied decimal point
5820J=0
5830hor=POS:ver=VPOS
5840REPEAT
5850G$=GET$
5860G=INSTR("0123456789",G$)
5870IF G>0 THEN PRINT G$;:st$=st$+G$:J=J+1
5880UNTIL J=before OR ASC(G$)=13 OR G$="." OR G$=CHR$(127)
5890IF G$=CHR$(127) THEN PRINT TAB(hor,ver) LEFT$(st$,(LEN(st$)-1));:st$=LEFT$(st$,(LEN(st$)-1)
):J=J-1:GOTO 5840
5900d=before-LEN(st$)
5910IF d>0 THEN st$=STRING$(d,"0")+st$+".":PRINT TAB(hor,ver) st$;
5920IF after=0 THEN field_cont$(I)=LEFT$(st$,before)+RIGHT$(st$,after):PRINT TAB(hor,ver) field
_cont$(I)+":":ENDPROC
5930IF J=before THEN PRINT".";
```

```
5940J=0
5950REPEAT
5960G$=GET$
5970G=INSTR("0123456789",G$)
5980IF G>0 THEN PRINT G$;:st$=st$+G$:J=J+1
5990UNTIL J=after OR ASC(G$)=13 OR G$=CHR$(127)
6000IF G$=CHR$(127) THEN PRINT TAB(hor,ver) LEFT$(st$,LEN(st$)-1);:st$=LEFT$(st$,LEN(st$)-1):J=
J-1:GOTO 5950
6010d=before+after-LEN(st$)
6020IF d>0 THEN st$=st$+STRING$(d,"0")
6030REM
6040field_cont$(I)=LEFT$(st$,before)+RIGHT$(st$,after)
6050ENDPROC
6060
6070
6080DEF PROCget_chars
6090REM Get character for the field of currently selected record.
6100LOCAL J,chars
6110field_cont$(I)=""
6120chars=VAL(width$(I))
6130FOR J=1 TO chars
6140G$=GET$
6150IF G$=CHR$(127) THEN J=J-2
6160 IF ASC(G$)=13 THEN field_cont$(I)=FNmodify(field_cont$(I),chars):J=chars ELSE PRINT G$;:fi
eld_cont$(I)=field_cont$(I)+G$
6170NEXT J
6180ENDPROC
6190
6200
6210DEF PROCshow_field
6220REM Display a selcted field of a record.
6230com$=command_stack$(2)
6240IF com$="" THEN PRINT "SYNTAX ERROR"ELSE PROCshow(VAL(command_stack$(2) ),VAL(command_stack
$(3)))
6250ENDPROC
6260DEF PROCshow(rec,field)
6270PRINT"field(";field;")";" of record(";rec;")=";FNget_field(rec,field)
6280ENDPROC
6290
6300
6310DEF PROCsort
6320REM Sort the index array using bubble sort.
6330LOCAL I,last,lastone_moved_down
6340last=VAL(nofrec$)
6350REPEAT
6360lastone_moved_down=0
6370FOR I=2 TO last
6380IF index$(I,1)<index$((I-1),1) THEN PROCswap(I,(I-1)):lastone_moved_down=I-1
6390NEXT I
6400last=lastone_moved_down
6410UNTIL last<2
6420FOR I=1 TO VAL(nofrec$)
6430PRINT£index_ch,index$(I,1)
6440PRINT£index_ch,index$(I,2)
6450NEXT I
6460CLOSE£index_ch
6470ENDPROC
6480
6490
6500DEF PROCswap(I,J)
6510REM Swap the content of two entries in the index array table.
```

```
6520LOCAL temp1$,temp2$
6530 temp1$=index$(I,1):temp2$=index$(I,2)
6540index$(I,1)=index$(J,1)
6550index$(I,2)=index$(J,2)
6560index$(J,1)=temp1$
6570index$(J,2)=temp2$
6580ENDPROC
>
```

```
L.
  10REM.................................MASTER ..............................
  20REM This is the main program. This program must first be executed prioir to the commencemen
t of any activity in the FMS.
  30MODE 7:OSWORD=&FFF1:OSBYTE=&FFF4:*FX 15,0
  40REM Define memory locations for dynamic databases (i.e. CAMP) and CellTalk. Allocate a name
 to each location.
  60HIMEM=&7B00:M=&7B00:logi_assoc=M+0:client_st_no=M+1:client_port=M+2:server_resp_mes=M+3:mes
s_stat=M+4:prog_stat=M+5:in_partbuf=M+6:out_partbuf=M+7:production_period=M+8:upload_prep=M+9:pa
rt_trans=M+10:where_part_trans=M+11
  70server_resp_part=M+12:robot_stat=M+13:trolley_stat=M+14:device_stat=M+15
  80REM Define the contents of these memory locations.
  90warehouse%=20:ready=&FF:not_ready=&00:nothing=&09:busy=&FF:free=&00:connect=&FF:disconnect=
&00:processed=&00:processing=&FF:not_recognised=&01:run_prog=&FF:running=&FF:stop=&00:download_s
ucc=&01:transfer_part=&FF:transfer_part_comp=&00
 100completed=&00:idle=&01:stopped=&02:critical_op_in_progress=&03:device_reset=&04:device_inop
erable=&05:uncorrectable_error_det=&06:correctable_error_det=&07:diagnostic_running=&08
 110started=&FF:in_progress=&FF:not_started=nothing
 130REM set the size of input and output part buffers.
 140from_trolley=&01:to_trolley=&02:full=2
 150REM Assign a key to each field in the system static data.
 160op_TYPE=1:key1=op_TYPE:CC_NAME=2:key2=CC_NAME:CC_CON=3:key3=CC_CON:STAT_NO=4:key4=STAT_NO:N
OF_PP=5:key5=NOF_PP:PP_NAME=6:key6=PP_NAME:SEQ_NO=7:key7=SEQ_NO:SETUP_T=8:key8=SETUP_T:DOWN_T=9:
key9=DOWN_T
 170proces_T=10:key10=proces_T:defect_R=11:key11=defect_R
 180REM If the order of fields for a job definition is changed, then adjust the keys
 190REM To economise the disc space usage assign the minimum desired number of records.
 200CLS:max_recs=6
 210REM Assign one record per production cell.
 220rec_fields=12
 230FOR I=0 TO 16:M?I=nothing:NEXT I
 240A%=0:max_peek_length%=5:max_poke_length%=5:current_rec=1
 250DIM cblock%40,peekbuffer% max_peek_length% ,pokebuffer% max_poke_length%
 260DIM field_cont$(rec_fields),index$(max_recs,2),field$(rec_fields),type$(rec_fields),width$(
rec_fields),dp$(rec_fields),command_stack$(8)
 270CLS:PROCfirst_menu
 280END
 290
 300
 310REM Place the manufacturing database routine here.
 320
 330 The communication module resides here.
 340
 350
 360DEF PROCfirst_menu
 370REM Provide the user with a menu.
 380LOCAL menu
 390menu=FALSE:REPEAT
 400IF D%=6 THEN reply=D%:D%=0:GOTO 560
 410IF D%=8 THEN reply=D%:D%=0:GOTO 560
 420CLS:PRINT TAB(0,3)"The following options are available."
 430PRINT TAB(0,4)"------------------------------------"
 440PRINT'"_ <1>  Define a new job."
 450PRINT"_ <2>  Create a job similar to an existing one."
 460PRINT"_ <3>  Modify an existing job."
 470PRINT"_ <4>  Process a job."
 480PRINT"_ <5>  FMS cell requirement check."
 490PRINT"_ <6>  Give historical report on FMS cells."
 500PRINT"_ <7>  Enter database utilities."
 510PRINT"_ <8>  Prepare an upload file to collect information after a production period."
 520PRINT"_ <9>  Show historical system_wide scheduling chart."
```

```
  530PRINT"_ <10> Quit."
  540REPEAT:INPUT TAB(0,22)"Your choice: "reply:UNTIL reply>0 AND reply<11
  550B%=reply
  560IF reply=1 THEN CLS:PRINT TAB(0,23) STRINGS(30," "):PRINT TAB(1,23) "Wait...":CHAIN"MASTER_
S1"
  570IF reply=2 THEN CLS:PRINT TAB(0,23) STRINGS(30," "):PRINT TAB(1,23) "Wait...":CHAIN"MASTER_
S1"
  580IF reply=3 THEN CLS:PRINT TAB(0,23) STRINGS(30," "):PRINT TAB(1,23) "Wait...":CHAIN"MASTER_
S1"
  590IF reply=4 THEN PRINT TAB(0,23) STRINGS(30," "):PRINT TAB(1,23) "Wait...":CHAIN"SCHEDULER"
  600IF reply=5 THEN PRINT TAB(0,23) STRINGS(30," "):PRINT TAB(1,23) "Wait...":CHAIN"MASTER_S1"
  610IF reply=6 THEN PROChistorical_rep
  620IF reply=7 THEN CLS:PRINT TAB(0,23) STRINGS(30," "):PRINT TAB(1,23) "Wait...":CHAIN"MASTER_
S1"
  630IF reply=8 THEN PROCprepare_upload
  640IF reply=9 THEN PRINT TAB(0,23) STRINGS(30," "):PRINT TAB(1,23) "Wait...":CHAIN"ACT_CHART"
  650IF reply=10 THEN menu=TRUE
  660REMA%=0
  670UNTIL menu
  680ENDPROC
  690
  700
  710DEF PROCthird_menu
  720LOCAL reply
  730CLS:PRINT TAB(0,3)"The following options are available."
  740PRINT"------------------------------------"
  750PRINT'"_ <1> Edit an existing job."
  760PRINT"_ <2> Add record to an existing job."
  770
  780REPEAT:INPUT TAB(0,20)"Your choice: "reply:UNTIL reply>0 AND reply<3
  790CLS:PRINT TAB(5,11):INPUT"Enter the name of the job :"command_stackS(3):command_stackS(2)="
UPDATE":command_stack$(5)=command_stackS(3)+"_IND":PROCopen_update_file:keyfieldS="1":index_ch=O
PENUP(command_stackS(5))
  800IF reply=1 THEN CLS:PROClist("TRUE"):INPUT'"Enter the number of the record to be edited :"r
ec_no:PROCedit(rec_no):CLS:PROClist("TRUE")
  810IF reply=2 THEN PROCadd_rec
  820PRINT TAB(0,23) STRINGS(30," "):PRINT TAB(1,23) "Wait...":PRINT£index_ch,keyfield$
  830CLOSE£index_ch:keyfieldS="1":PROCorder(command_stackS(5),keyfieldS)
  840mod_idx$=command_stack$(3)+"_INS":keyfieldS="7":PROCorder(mod_idxS,keyfieldS):CLOSE£db_ch

  850
  860
  870DEF PROCprepare_upload
  880REM Issue commands to the active cells to upload a report on part production in their cells
. Create an upload file to store the information for historical viewing. Update the upload file
if it already exists.
  890LOCAL I,K,temp_ch%
  900CLS:PRINT TAB(5,11):INPUT"Enter the name of the job which has been currently under process
:"job_name$:command_stack$(2)=job_name$:command_stack$(3)="INDEX":command_stackS(4)=job_nameS+"_
INS":PROCopen_file
  910nof_active_cell%=0
  920FOR I=1 TO VAL(nofrec$)
  930IF indexS(I,1)<>"00" THEN nof_active_cell%=nof_active_cell%+1
  940NEXT I
  950DIM upload_mes$(nof_active_cell%)
  960K=1
  970FOR I=1 TO VAL(nofrecS)
  980IF indexS(I,1)<>"00" THEN s%=FNget_field(VAL(indexS(I,2)),key4):loc%=upload_prep:pokebuffer
%?0=in_progress:PROCpoke(s%,loc%,1) ELSE NEXT I
  990REPEAT:UNTIL ?logi_associ=connect:?server_resp_mes=ready:PROCget_mail:?server_resp_mes=not_
ready
```

```
 1000PRINT"The message from station ";s%;" is :";upload_mes$(K):K=K+1
 1010NEXT I
 1020PROCclose(job_name$):PROCdelay(300)
 1030PRINT"Preparing cells historical information.":PRINT TAB(0,23) STRING$(39," "):PRINT TAB(1,
23) "Wait..."
 1040up_file$=job_name$+"U":up_idx$=up_file$+"_IND"
 1050temp_ch%=OPENUP(up_file$)
 1060IF temp_ch%<>0 THEN CLOSE£temp_ch%
 1070IF temp_ch%=0 THEN PROCcreate_upload_file(up_file$,up_idx$)
 1080PROCopen_update(up_file$)
 1090IF temp_ch%<>0 THEN index_ch=OPENUP(up_idx$)
 1100PROCadd:keyfield$="1":PRINT£index_ch,keyfield$:CLOSE£index_ch:command_stack$(5)=up_idx$:PRO
Corder(command_stack$(5),keyfield$):CLOSE£db_ch
 1110ENDPROC
 1120
 1130
 1140DEF PROChistorical_rep
 1150REM Provide a historical report of a job.
 1160LOCAL I,K,total,defect
 1170total=0:defect=0
 1180MODE 0
 1190CLS:PRINT TAB(5,11):INPUT"Enter the name of an existing job :"job_name$:command_stack$(2)=j
ob_name$+"U":command_stack$(3)="INDEX":command_stack$(4)=job_name$+"U_IND":PROCopen_file
 1200CLS:PRINT"               ***** Historical report on FMS cells *****"
 1210PRINT'"               Total number of batches for ";job_name$;" to date : ";VAL(nofrec$)
 1220FOR I=1 TO VAL(nofrec$):rec=VAL(index$(I,2))
 1230PRINT'"--------Uploaded information on part production of batch no. ";rec;" ---------"
 1240FOR K=1 TO n
 1250field_con$=FNget_field(rec,K)
 1260IF MID$(field_con$,4,1)=" " THEN cell_name$=LEFT$(field_con$,3)+"ING" ELSE cell_name$=LEFT$
(field_con$,4)+"ING"
 1270PRINT'"     <<<< ";cell_name$+" cell >>>>     "
 1280PRINT"Number of processed components   : ";MID$(field_con$,6,2)
 1290IF K=1 THEN total=VAL(MID$(field_con$,6,2))
 1300PRINT"Number of defective components   : ";MID$(field_con$,9,2):defect=VAL(MID$(field_con$,
9,2))+defect
 1310PRINT"Defect rate of the cell          : ";MID$(field_con$,12,5);" %"
 1320PRINT"Efficiency of the cell           : ";MID$(field_con$,18,5);" %"
 1330NEXT K
 1340good=VAL(MID$(field_con$,6,2))-VAL(MID$(field_con$,9,2))
 1350PRINT'"..... Number of raw components required for this batch     : ";total;" ....."
 1360PRINT"..... Number of good components manufactured in this batch : ";good;" ....."
 1370total=0:defect=0:good=0
 1380g$=GET$
 1390IF g$="P" THEN PROCprint
 1400CLS:NEXT I
 1410PRINT"Press any key to get back to the menu.":g=GET
 1420CLOSE£0
 1430MODE 7
 1440ENDPROC
 1450
 1460
 1470DEF PROCprint
 1480*SCDUMP
 1490ENDPROC
>
```

L.

```
  10REM............................ MASTER_S1 ................................
  20REM This is an associated program of MASTER.
  30MODE 7:OSWORD=&FFF1:OSBYTE=&FFF4:*FX 15,0
  40HIMEM=&7B00:M=&7B00
  50op_TYPE=1:key1=op_TYPE:CC_NAME=2:key2=CC_NAME:CC_CON=3:key3=CC_CON:STAT_NO=4:key4=STAT_NO:N
OF_PP=5:key5=NOF_PP:PP_NAME=6:key6=PP_NAME:SEQ_NO=7:key7=SEQ_NO:SETUP_T=8:key8=SETUP_T:DOWN_T=9:
key9=DOWN_T
  60proces_T=10:key10=proces_T:defect_R=11:key11=defect_R
  70REM If the order of fields for a job definition is changed, then adjust the variable keys a
ccordingly.
  80CLS:max_recs=6:REM Maximum number of records, a record for each cell.
  90rec_fields=12:REM Program can handle 12 fields per record
 100max_peek_length%=5:max_poke_length%=5:current_rec=1
 110DIM field_cont$(rec_fields),index$(max_recs,2),field$(rec_fields),type$(rec_fields),width$(
rec_fields),dp$(rec_fields),command_stack$(8)
 130END
 140
 150
 160REM The manufacturing database routine is placed here.
 170REM The communication module resides here.
 180
 190
 200DEF PROCseq_active_cell_check
 210REM Check whether the cells defined in the part production route are active.
 220LOCAL I
 230FOR I=1 TO VAL(nofrec$)
 240IF index$(I,1)="00" THEN IF LEFT$(FNget_field(VAL(index$(I,2)),key3),1)="A" THEN PRINT "INV
ALID ACTIVE CELL IN RECORD ";VAL(index$(I,2)):ENDPROC
 250IF index$(I,1)<>"00" THEN IF LEFT$(FNget_field(VAL(index$(I,2)),key3),1)="I" THEN PRINT "IN
VALID SEQUENCE IN RECORD =";VAL(index$(I,2)):ENDPROC
 260NEXT I
 270ENDPROC
 280
 290
 300DEF PROCfms_cell_requirement_check
 310REM Provide a FMS cell requirement check.
 320LOCAL good_component,analyzed_period,shift
 330@%=&20209
 340MODE 0
 350CLS:PRINT TAB(5,11):INPUT"Enter the job name: "command_stack$(2):job_name$=command_stack$(2
)
 360INPUT"Enter the number of parts to be manufactured: "good_component
 370INPUT"Enter the analyzed period in hours: "analyzed_period
 380INPUT"Enter the length of production period(i.e. shift) in hours: "shift
 390command_stack$(3)="INDEX":command_stack$(4)=command_stack$(2)+"_INS":PROCopen_file
 400CLS:PRINT'"Press RETURN and wait to proceed":g=GET
 410FOR I=1 TO VAL(nofrec$)
 420IF index$(I,1)="00" THEN NEXT I
 430restof_recs%=I
 440DIM raw_p(VAL(nofrec$))
 450good_comp=good_component
 460FOR K=VAL(nofrec$) TO restof_recs%  STEP -1
 470rec=VAL(index$(K,2)):defcet_rate=FNget_field(rec,key11)/100:raw_p(K)=good_comp*defcet_rate+
good_comp
 480good_comp=raw_p(K)
 490NEXT K
 500FOR J=restof_recs% TO VAL(nofrec$):rec=VAL(index$(J,2))
 510CLS:PRINT TAB(22,1)"***** FMS cell capacity check for ";job_name$;" *****"
 520REM Define the mathematical model for the cell capacity check.
 530defect_rate=FNget_field(rec,key11):av_setup_time=FNget_field(rec,key8):av_down_time=FNget_f
```

```
ield(rec,key9):processing_time=FNget_field(rec,key10):station$=FNget_field(rec,key1)
  540output_rate=good_component/(1-(defect_rate/100)):efficiency=1-(av_setup_time+av_down_time)/
(analyzed_period *60):requirement=(processing_time*output_rate)/(60*shift*efficiency)
  550PRINT'"            ***** ";station$;"cell";" *****"
  560PRINT"Input data"
  570PRINT"----------"
  580PRINT"Finished part required per production period  =";good_component
  590PRINT"Length of production period                   =";shift;" Hours"
  600PRINT"Analyzed period of time                       =";analyzed_period;" Hours"
  610PRINT"Average setup time                            =";av_setup_time;" Mins"
  620PRINT"Average downtime                              =";av_down_time;" Mins"
  630PRINT"Processing time per component                 =";processing_time;" Mins"
  640PRINT"Defect rate of the cell                       =";defect_rate;" %"
  650PRINT'"Calculated results"
  660PRINT"------------------"
  670REMPRINT"Raw parts required per production period      =";raw_p(J)
  680U%=raw_p(J):X=0
  690REPEAT:X=X+1:UNTIL MID$(STR$(raw_p(J)),X,1)="."
  700IF VAL(MID$(STR$(raw_p(J)),X+1,1))<5 THEN raw_p(J)=U%:PRINT"Raw parts required per producti
on period      =";raw_p(J)
  710IF VAL(MID$(STR$(raw_p(J)),X+1,1))>5 OR VAL(MID$(STR$(raw_p(J)),X+1,1))=5 THEN raw_p(J)=U%+
1:PRINT"Raw parts required per production period      =";raw_p(J)
  720PRINT"FMS cell efficiency                           =";efficiency*100;" %"
  730PRINT"Unadjusted equipment requirement              =";requirement;" FMS cells"
  740PRINT"Adjusted equipment requirement                =";"1.00";" FMS cells"
  750g$=GET$
  760IF g$="P" THEN PROCprint
  770NEXT J
  780CLOSE£0:@%=10:g=GET
  790ENDPROC
  800
  810
  820DEF PROCcreate_upload_file
  830REM Create an upload file to store the reports provided by the cell controllers on the part
 production in their cells.
  840LOCAL temp_ch
  850CLS:PRINT TAB(5,11):INPUT"Enter the job name :"command_stack$(2):PRINT TAB(0,23) STRING$(30
," "):PRINT TAB(1,23) "Wait...":db_file$=command_stack$(2)+"U":mod_idx$=db_file$+"_IND"
  860temp_ch=OPENUP(db_file$)
  870IF temp_ch<>0 THEN CLOSE£temp_ch:CHAIN"SCHEDULER"
  880
  890
  900DEF PROCprint
  910*SCDUMP
  920ENDPROC
>
```

L.

```
   10REM...........................MILL_VMD ...........................
   20REM This is a cell controller specific software for the milling cell. To use this program f
 or another cell for example sawing cell, all the control signals for the specific contoller have
  to be replaced in the appropriate procedures.
   30MODE 7:OSWORD=&FFF1:OSBYTE=&FFF4:*FX 15,0
   40REM Assign a name to each memory location (this is the building block of dynamic database).

   50HIMEM=&7B00:M=&7B00:logi_assoc=M+0:client_st_no=M+1:client_port=M+2:server_resp_mes=M+3:mes
 s_stat=M+4:prog_stat=M+5:in_partbuf=M+6:out_partbuf=M+7:production_period=M+8:upload_prep=M+9:pa
 rt_trans=M+10:where_part_trans=M+11
   60server_resp_part=M+12:robot_stat=M+13:trolley_stat=M+14:device_stat=M+15:defect_part_stat=M
+16:nof_defect_part=M+17:total_nof_parts=M+18:nof_processed_parts=M+19
   70warehouse%=20:ready=&FF:not_ready=&00:nothing=&09:busy=&FF:free=&00:connect=&FF:disconnect=
&00:processed=&00:processing=&FF:not_recognised=&01:run_prog=&FF:running=&FF:stop=&00:download_s
ucc=&01:transfer_part=&FF:transfer_part_comp=&00
   80empty=&00:completed=&00:idle=&01:stopped=&02:critical_op_in_progress=&03:device_reset=&04:d
evice_inoperable=&05:uncorrectable_error_det=&06:correctable_error_det=&07:diagnostic_running=&0
8
   90defect=&FF:in_progress=&FF:not_started=nothing:started=&FF
  100from_trolley=&01:to_trolley=&02:full=2
  110robot_st%=14:this_cell$="MILLING"
  120op_TYPE=1:key1=op_TYPE:CC_NAME=2:key2=CC_NAME:CC_CON=3:key3=CC_CON:STAT_NO=4:key4=STAT_NO:N
OF_PP=5:key5=NOF_PP:PP_NAME=6:key6=PP_NAME:SEQ_NO=7:key7=SEQ_NO
  130path1=1:path2=2:path3=3:path4=4
  140FOR I=0 TO 30:M?I=nothing:NEXT I
  150?in_partbuf=0:?out_partbuf=0:?nof_defect_part=0:?total_nof_parts=0:?nof_processed_parts=0
  160nof_processed_parts%=0:nof_parts_tobe_manufactured%=0:Part_processing_time%=0:setup_time%=0

  170max_recs=6:rec_fields=11
  180max_peek_length%=5:max_poke_length%=5:current_rec=1
  190DIM cblock%40,peekbuffer% max_peek_length% ,pokebuffer% max_poke_length%
  200DIM field_cont$(rec_fields),index$(max_recs,2),field$(rec_fields),type$(rec_fields),width$(
rec_fields),dp$(rec_fields),command_stack$(8)
  210quit=FALSE
  220REM Wait until proceed signal is given by the master.
  230REPEAT:UNTIL ?prog_stat=run_prog:?production_period=started:?upload_prep=not_started:PROCin
itialise:ok=FALSE:REPEAT
  240REPEAT:UNTIL ?logi_assoc=connect
  250?server_resp_mes=ready
  260REM Get the name of a job to be processed.
  270PROCget_mail:?server_resp_mes=not_ready:master_st%=?client_st_no
  280UNTIL ok
  290?mess_stat=nothing:PROCinit_robot
  300REM Search the database for the relevant information on the current job.
  310command_stack$(2)=job_name$:command_stack$(3)="INDEX":command_stack$(4)=job_name$+"_IND":PR
OCopen_file
  320FOR I=1 TO VAL(nofrec$)
  330IF LEFT$(index$(I,1),7)<>this_cell$ THEN NEXT I
  340this_rec%=VAL(index$(I,2)):PRINT"Record ";this_rec%;" is allocated to this cell"
  350this_st%=FNget_field(this_rec%,key4)
  360PRINT"This is st. '";this_st%;"'"
  370part_p$=FNget_field(this_rec%,key6)
  380PRINT"Down loading part program ";part_p$
  390REM Reset the programmable devices in the cell before downloading the part program to thier
 controllers.
  400PROCreset:PROCdownload(part_p$)
  410command_stack$(2)=job_name$:PROCclose_file
  420command_stack$(2)=job_name$:command_stack$(3)="INDEX":command_stack$(4)=job_name$+"_INS":PR
OCopen_file
  430FOR I=1 TO VAL(nofrec$)
```

```
440REM Find the name and network address of the adjacent logical cells.
450IF index$(I,2)<>STR$(this_rec%) THEN NEXT I
460seq=VAL(index$(I,1))
470IF seq<>1 THEN prvious_st%=FNget_field(VAL(index$((I-1),2)),key4) ELSE prvious_st%=0
480PRINT'"previous logical cell is (st. '";prvious_st%;"')"
490REPEAT
500IF ?prog_stat=stopped THEN PROCterminate_part_prod
510UNTIL ?part_trans=transfer_part
520?server_resp_part=ready
530REM Get part specific data directly from the cell which currently has dispatched this part.

540IF prvious_st%<>0 THEN mes$="SEND PART ORIENTATION":s%=prvious_st%:PROCinfo_request(mes$,s%
)
550REM Receive a part from warehous or previous logical cell.
560PROCget_a_part
570PROCpart_transfer(path2):b=?in_partbuf:b=b-1:?in_partbuf=b
580REM Start the operation on the current part in the cell.
590PROCstart_milling
600PROCdelay(100)
610REPEAT
620IF ?logi_assoc=connect THEN ?server_resp_mes=ready:PROCget_mail:?server_resp_mes=not_ready

630REM Check wether the master has issued a command for part production to be terminated in th
is cell.
640IF ?prog_stat=stopped THEN PROCterminate_part_prod
650IF ?part_trans=transfer_part THEN ?server_resp_part=ready:REPEAT:UNTIL ?where_part_trans=fr
om_trolley OR ?where_part_trans=to_trolley
660IF ?prog_stat=stopped THEN PROCterminate_part_prod
670IF ?where_part_trans=from_trolley THEN IF prvious_st%<>0 THEN mes$="SEND PART ORIENTATION":
s%=prvious_st%:PROCinfo_request(mes$,s%):PROCget_a_part
680IF ?where_part_trans=from_trolley THEN IF prvious_st%=0 THEN PROCget_a_part
690IF ?where_part_trans=to_trolley THEN PROCsend_a_part
700REM Check whether the programmable device has beencompleted its operation on the part. If o
peration has completed successfully start operating on the next part.
710IF FNmill_monitor THEN PROCprocess_next_part
720UNTIL nof_processed_parts%=nof_parts_tobe_manufactured%
730IF ?total_nof_parts<>nof_parts_tobe_manufactured% THEN nof_parts_tobe_manufactured%=?total_
nof_parts:GOTO 570
740REPEAT
750IF ?prog_stat=stopped THEN PROCterminate_part_prod
760IF ?logi_assoc=connect THEN ?server_resp_mes=ready:PROCget_mail:?server_resp_mes=not_ready

770UNTIL ?part_trans=transfer_part
780?server_resp_part=ready
790REPEAT
800IF ?prog_stat=stopped THEN PROCterminate_part_prod
810IF ?logi_assoc=connect THEN ?server_resp_mes=ready:PROCget_mail:?server_resp_mes=not_ready

820UNTIL ?where_part_trans=to_trolley OR ?where_part_trans=from_trolley
830IF ?where_part_trans=to_trolley THEN PROCsend_a_part
840IF ?where_part_trans=from_trolley THEN IF prvious_st%<>0 THEN mes$="SEND PART ORIENTATION":
s%=prvious_st%:PROCinfo_request(mes$,s%):nof_parts_tobe_manufactured%=nof_parts_tobe_manufacture
d%+1:GOTO 560
850IF ?where_part_trans=from_trolley THEN IF prvious_st%=0 THEN nof_parts_tobe_manufactured%=n
of_parts_tobe_manufactured%+1:GOTO 560
860REPEAT
870IF ?production_period<>completed THEN IF ?part_trans=transfer_part THEN ?server_resp_part=r
eady
880IF ?where_part_trans=from_trolley THEN IF prvious_st%<>0 THEN mes$="SEND PART ORIENTATION":
s%=prvious_st%:PROCinfo_request(mes$,s%):nof_parts_tobe_manufactured%=nof_parts_tobe_manufacture
d%+1:GOTO 560
```

```
890IF ?where_part_trans=from_trolley THEN IF prvious_st%=0 THEN nof_parts_tobe_manufactured%=n
of_parts_tobe_manufactured%+1:GOTO 560
900IF ?where_part_trans=to_trolley THEN PROCsend_a_part
910IF ?prog_stat=stopped THEN PROCterminate_part_prod
920IF ?logi_assoc=connect THEN ?server_resp_mes=ready:PROCget_mail:?server_resp_mes=not_ready
930UNTIL ?production_period=completed
940command_stack$(2)=job_name$:PROCclose_file:?device_stat=completed
950REM Send a report on the current part production to the master.
960PROCcell_reports
970REPEAT:UNTIL ?upload_prep=in_progress
980lower=15:upper=40:nof_good_comp$=STR$(FNrnd_num(lower,upper))
990lower=1:upper=2:nof_defective_comp$=STR$(FNrnd_num(lower,upper))
1000lower=92:upper=99:first$=STR$(FNrnd_num(lower,upper)):lower=10:upper=99:second$=STR$(FNrnd_
num(lower,upper)):efficiency$=first$+"."+second$
1010@%=&01020209
1020total_parts=VAL(nof_good_comp$)+VAL(nof_defective_comp$)
1030defect_rate=VAL(nof_defective_comp$)/total_parts
1040defect_rate$=STR$(defect_rate):@%=10:IF LEN(defect_rate$)=4 THEN defect_rate$=defect_rate$+
" "
1050mes$=LEFT$(this_cell$,4)+"_"+nof_good_comp$+"_"+nof_defective_comp$+" _"+defect_rate$+"_"+e
fficiency$
1060PROCtx_message(this_st%,master_st%,mes$)
1070?upload_prep=completed
1080?prog_stat=completed:PRINT"Cell info :";mes$
1090END
1100
1110
1120REM The manufacturing database resides here.
1130
1140REM The communication routine (CellTalk) is placed here.
1150
1160
1170DEF PROCrespond
1180REM Send a reply to the requester cell. All the questions which are likely to be asked have
to be defined here for the cell contoller to be able to recognise them.
1190client_st%=?client_st_no
1200IF MID$(mess$,6)="PART ORIENTATION" THEN PROCget_part_orientation:PROCtx_message(this_st%,c
lient_st%,mes$):ENDPROC
1210IF MID$(mess$,6)="PART DIMENTION" THEN PROCget_part_dimention:PROCtx_message(this_st%,clien
t_st%,mes$):ENDPROC
1220PRINT"ROUTINE IS NOT DEFINED !."
1230ENDPROC
1240
1250
1260DEF PROCinfo_request(query$,server_st%)
1270REM Send a request to another cell for their cell specific data.
1280server=FALSE
1290PROCtx_message(this_st%,server_st%,query$)
1300REPEAT
1310IF ?logi_assoc=connect THEN IF ?client_st_no=server_st% THEN server=TRUE:?server_resp_mes=r
eady:PROCget_mail:?server_resp_mes=not_ready
1320IF ?logi_assoc=connect THEN IF ?client_st_no<>server_st% THEN ?server_resp_mes=ready:PROCge
t_mail:?server_resp_mes=not_ready
1330IF ?prog_stat=stopped THEN PROCterminate_part_prod
1340UNTIL server
1350ENDPROC
1360
1370
1380DEF PROCget_part_orientation
1390REM This is an example of the cell specific data.
1400lower=10:upper=99:X$=STR$(FNrnd_num(lower,upper)):lower=10:upper=99:Y$=STR$(FNrnd_num(lower
,upper)):lower=10:upper=99:deg$=STR$(FNrnd_num(lower,upper)):lower=10:upper=99:min$=STR$(FNrnd_n
um(lower,upper))
 1410mes$="O"+"XOO"+X$+"YOO"+Y$+"ANG"+deg$+"."+min$
 1420ENDPROC
 1430
 1440
 1450DEF PROCget_part_dimention
 1460REM This is an example of the cell specific data.
 1470lower=10:upper=99:L$=STR$(FNrnd_num(lower,upper)):lower=10:upper=99:W$=STR$(FNrnd_num(lower
,upper)):lower=10:upper=99:H$=STR$(FNrnd_num(lower,upper))
 1480mes$="D"+"LOO"+L$+"WOO"+W$+"HOO"+H$
 1490ENDPROC
 1500
 1510
 1520DEF PROCdelay(n%)
 1530REM wait for n% centiseconds
 1540LOCAL limit%
 1550limit%=TIME+n%:REPEAT:UNTIL TIME>=limit%
 1560ENDPROC
 1570
 1580
 1590DEF PROCdownload(part_prog$)
 1600REM Download part program to the controller of the specific programmable device controller.
 1610LOCAL X,S%,K%,J,N%
 1620PROCcontrol(129):PROCcontrol(178)
 1630PRINT''
 1640PRINT"Down loading ";part_prog$;"part program into the mill via serial port.":PRINT''
 1650REM Get the part program from disk.
 1660X=OPENIN(part_prog$)
 1670*FX2,2
 1680*FX3,5
 1690*FX8,7
 1700*FX15,0
 1710A%=&9C:X%=&00:Y%=&E3
 1720CALL &FFF4
 1730REPEAT:INPUT£X,table_code$:FOR I=1 TO LEN(table_code$):PRINT MID$(table_code$,I,1);:PROCdel
ay(13):NEXT I:UNTIL table_code$="END"
 1740*FX2,0
 1750*FX3,0
 1760*FX4,0
 1770CLOSE£X
 1780ENDPROC
 1790
 1800
 1810DEF PROCstart_milling
 1820REM Send the start code to the miller.
 1830?device_stat=critical_op_in_progress
 1840PROCmill_home_pos:PROCcontrol(6)
 1850PROCcontrol(139):PROCcontrol(48):PROCcontrol(48):PROCcontrol(177):TIME=0
 1860PRINT"@@@@ Milling operation in progress @@@@"
 1870PROCdelay(100)
 1880ENDPROC
 1890
 1900
 1910DEF PROCsend(M%)
 1920REM  Send data through the serial port.
 1930PROCdelay(13):*FX8,7
 1940*FX15,0
 1950*FX3,5
 1960*FX2,2
 1970A%=&9C:Y%=&E3:X%=&08:CALL &FFF4
 1980PRINT CHR$(M%);:*FX3,0
 1990*FX2,0
 2000ENDPROC
 2010
 2020
 2030DEF PROCinitialise
 2040REM Initialise the relavent port on the cell interface module (i.e. port B of VIA1 as outpu
t).
 2050A%=&93:X%=&E2:Y%=&FF:CALL OSBYTE
 2060ENDPROC
 2070
 2080
 2090DEF PROCcontrol(value%)
 2100REM Send Control command to the controller of specific device.
 2110A%=&93:X%=&EC:Y%=&C0:CALL OSBYTE
 2120A%=&93:X%=&E0:Y%=value% EOR &FF:CALL OSBYTE
 2130A%=&93:X%=&EC:Y%=&E0:CALL OSBYTE
 2140A%=&93:X%=&EC:Y%=&C0:CALL OSBYTE
 2150ENDPROC
 2160
 2170
 2180DEF PROCreset
 2190REM Reset the device.
 2200?device_stat=device_reset
 2210REM Set port A on VIA1 as output.
 2220A%=&93:X%=&E3:Y%=&FF:CALL OSBYTE
 2230A%=&93:X%=&EC:Y%=&0E:CALL OSBYTE
 2240A%=&93:X%=&EC:Y%=&0C:CALL OSBYTE
 2250A%=&93:X%=&EC:Y%=&0E:CALL OSBYTE
 2260PROCmill_home_pos:PROCcontrol(6)
 2270?device_stat=idle
 2280ENDPROC
 2290
 2300
 2310DEF PROCmill_home_pos
 2320REM Send the mill's table to the home position.
 2330A%=&97:X%=&62:Y%=&F0:CALL OSBYTE
 2340A%=&97:X%=&60:Y%=&F0:CALL OSBYTE
 2350PROCdrive_xmill:PROCdrive_ymill
 2360ENDPROC
 2370DEF PROCdrive_xmill
 2380REM Drive the table motors remotely.
 2390LOCAL limit%
 2400limit%=FALSE:x_axis%=0:REPEAT
 2410A%=&96:X%=&60:yreg%=((USR(OSBYTE) AND &FF0000) DIV &10000)
 2420IF (&01 AND yreg%)=&01 THEN PRINT"Limit is reached on the x axis.":limit%=TRUE:PROCchannel_
reset:GOTO 2460
 2430A%=&97:X%=&60:Y%=&E0:CALL OSBYTE
 2440x_axis%=x_axis%+1
 2450PRINT TAB(5,10)"Travel on x axis is ";x_axis%
 2460UNTIL limit%
 2470ENDPROC
 2480
 2490
 2500DEF PROCdrive_ymill
 2510REM Drive the table motors remotely.
 2520LOCAL limit%
 2530limit%=FALSE:y_axis%=0:REPEAT
 2540A%=&96:X%=&60:yreg%=((USR(OSBYTE) AND &FF0000) DIV &10000)
 2550IF (&02 AND yreg%)=&02 THEN PRINT"Limit is reached on the y axis.":limit%=TRUE:PROCchannel_
```

```
reset:GOTO 2590
 2560A%=&97:X%=&60:Y%=&D0:CALL OSBYTE
 2570y_axis%=y_axis%+1
 2580PRINT TAB(5,13)"Travel on y axis is ";y_axis%
 2590UNTIL limit%
 2600ENDPROC
 2610
 2620
 2630DEF FNmill_monitor
 2640REM Monitor whether or not the milling operation has completed. More monitoring functions c
ould be built here.
 2650op_complete=FALSE
 2660A%=&96:X%=&60:yreg%=((USR(OSBYTE) AND &FF0000) DIV &10000)
 2670IF (&01 AND yreg%)=&01 AND (&02 AND yreg%)=&02 THEN op_complete=TRUE:nof_processed_parts%=n
of_processed_parts%+1:PRINT"+++ Milling operation is completed +++":?device_stat=completed:b=?no
f_processed_parts:b=b+1:?nof_processed_parts=b
 2680IF (&01 AND yreg%)=&01 AND (&02 AND yreg%)=&02 THEN IF nof_processed_parts%=1 OR nof_proces
sed_parts%=3 THEN defective_parts%=1:?defect_part_stat=defect:b=?nof_defect_part:b=b+1:?nof_defe
ct_part=b
 2690IF (&01 AND yreg%)=&01 AND (&02 AND yreg%)=&02 THEN IF nof_processed_parts%=1 OR nof_proces
sed_parts%=3 THEN ENVELOPE 2,1,2,-2,2,10,20,10,1,0,0,-1,100,100 :SOUND 1,2,100,100
 2700=op_complete
 2710
 2720
 2730DEF PROCchannel_reset
 2740A%=&97:X%=&60:Y%=&F0:CALL OSBYTE
 2750ENDPROC
 2760
 2770
 2780DEF PROCinit_robot
 2790REM Command to download the robot program resides here.
 2800PRINT"Robot program is downloaded now"
 2810ENDPROC
 2820
 2830
 2840DEF PROCpart_transfer(robot_path)
 2850REM Give a name to each pick and place operation of the robot.
 2860?robot_stat=busy:TIME=0
 2870IF robot_path=1 THEN PRINT"***Part transfer from trolley to input buffer in progress.**":PR
OCdelay(400)
 2880IF robot_path=2 THEN PRINT"***Part transfer from input buffer to machine's table in progres
s.**":PROCdelay(100):setup_time%=setup_time%+TIME
 2890IF robot_path=3 THEN IF defective_parts%=1 AND ?out_partbuf=empty THEN PRINT"***Defective p
art transfer from machine's table to output buffer in progress.**":PROCdelay(100)
 2900IF robot_path=3 THEN IF defective_parts%=0 AND ?defect_part_stat=nothing THEN PRINT"***Part
 transfer from machine's table to output buffer in progress.**":PROCdelay(100)
 2910IF robot_path=3 THEN IF defective_parts%=1 AND ?out_partbuf<>empty PROCsend_defective_part:
PRINT"***Part transfer from machine's table to output buffer in progress.**":PROCdelay(100)
 2920IF robot_path=3 THEN IF defective_parts%>1 AND ?out_partbuf<>empty PROCsend_defective_part
 2930IF robot_path=4 THEN PRINT"***Part transfer from output buffer to trolley in progress.**":P
ROCdelay(100)
 2940PRINT'"***** Part transfer completed. *****"
 2950?robot_stat=free
 2960ENDPROC
 2970
 2980
 2990DEF PROCprocess_next_part
 3000REM Prepare the cell or the production of the next part.
 3010REPEAT
 3020IF ?prog_stat=stopped THEN PROCterminate_part_prod
 3030IF ?logi_assoc=connect THEN ?server_resp_mes=ready:PROCget_mail:?server_resp_mes=not_ready
```

```
3040IF ?out_partbuf=full THEN PROCout_partbuf_full
3050UNTIL ?out_partbuf<>full
3060PROCpart_transfer(path3):b=?out_partbuf:b=b+1:?out_partbuf=b
3070IF nof_processed_parts%=nof_parts_tobe_manufactured% THEN ENDPROC
3080IF ?in_partbuf<>0 THEN PROCpart_transfer(path2):b=?in_partbuf:b=b-1:?in_partbuf=b:PROCstart
_milling:PROCdelay(100):ENDPROC
3090REPEAT
3100IF ?prog_stat=stopped THEN PROCterminate_part_prod
3110IF ?logi_associ=connect THEN ?server_resp_mes=ready:PROCget_mail:?server_resp_mes=not_ready

3120UNTIL ?part_trans=transfer_part:?server_resp_part=ready
3130IF ?where_part_trans=from_trolley THEN IF prvious_st%<>0 THEN mes$="SEND PART ORIENTATION":
s%=prvious_st%:PROCinfo_request(mes$,s%):PROCget_a_part:PROCpart_transfer(path2):b=?in_partbuf:b
=b-1:?in_partbuf=b:PROCstart_milling:ENDPROC
3140IF ?where_part_trans=from_trolley THEN IF prvious_st%=0 THEN PROCget_a_part:PROCpart_transf
er(path2):b=?in_partbuf:b=b-1:?in_partbuf=b:PROCstart_milling:ENDPROC
3150IF ?where_part_trans=to_trolley THEN PROCsend_a_part
3160GOTO 3090
3170ENDPROC
3180
3190
3200DEF PROCout_partbuf_full
3210REM Before placing a part onto the output part buffer check whether it is full or not.
3220REPEAT
3230IF ?prog_stat=stopped THEN PROCterminate_part_prod
3240IF ?logi_assoc=connect THEN ?server_resp_mes=ready:PROCget_mail:?server_resp_mes=not_ready
3250UNTIL ?part_trans=transfer_part:?server_resp_part=ready
3260IF ?where_part_trans=from_trolley THEN IF prvious_st%<>0 THEN mes$="SEND PART ORIENTATION":
s%=prvious_st%:PROCinfo_request(mes$,s%):PROCget_a_part
3270IF ?where_part_trans=from_trolley THEN IF prvious_st%=0 THEN PROCget_a_part
3280IF ?where_part_trans=to_trolley THEN PROCsend_a_part
3290ENDPROC
3300
3310
3320DEF PROCget_a_part
3330PROCpart_transfer(path1):b=?in_partbuf:b=b+1:?in_partbuf=b:?server_resp_part=not_ready:?par
t_trans=transfer_part_comp:?where_part_trans=nothing
3340b=?total_nof_parts:b=b+1:?total_nof_parts=b
3350s%=?client_st_no:loc%=?trolley_stat:pokebuffer%?0=free:PROCpoke(s%,loc%,1):PROCdelay(50)
3360ENDPROC
3370
3380
3390DEF PROCsend_a_part
3400PROCpart_transfer(path4):b=?out_partbuf:b=b-1:?out_partbuf=b:?server_resp_part=not_ready:?p
art_trans=transfer_part_comp:?where_part_trans=nothing
3410IF ?defect_part_stat=defect THEN defective_parts%=defective_parts%-1:PRINT"***Defective par
t transfer from output buffer to warehouse***"
3420s%=?client_st_no:loc%=?trolley_stat:pokebuffer%?0=free:PROCpoke(s%,loc%,1):PROCdelay(50)
3430ENDPROC
3440
3450
3460DEF PROCsend_defective_part
3470REPEAT
3480REPEAT
3490IF ?prog_stat=stopped THEN PROCterminate_part_prod
3500IF ?logi_associ=connect THEN ?server_resp_mes=ready:PROCget_mail:?server_resp_mes=not_ready

3510UNTIL ?part_trans=transfer_part:?server_resp_part=ready
3520IF ?where_part_trans=from_trolley THEN IF prvious_st%<>0 THEN mes$="SEND PART ORIENTATION":
s%=prvious_st%:PROCinfo_request(mes$,s%):PROCget_a_part
3530IF ?where_part_trans=to_trolley THEN PROCsend_a_part
```

```
3540UNTIL defective_parts%=0
3550ENDPROC
3560
3570
3580DEF FNrnd_num(lower_boundery,upper_boundery)
3590=lower_boundery-1+RND(upper_boundery-lower_boundery+1)
3600
3610
3620DEF PROCterminate_part_prod
3630PROCdelay(400)
3640IF ?device_stat=critical_op_in_progress THEN REPEAT:UNTIL FNmill_monitor
3650CLOSE£0
3660SOUND 1,-15,130,5:SOUND 1,-15,150,5:SOUND 1,-15,190,5:SOUND 1,-15,255,5
3670PRINT'"<<<< Terminating part production in >>>>"
3680PRINT"                ";this_cell$;" cell"
3690PRINT"                ------------"
3700PROCcell_reports
3710STOP
3720ENDPROC
3730REM--cell_reports--
3740DEF PROCcell_reports
3750REM Generate a report on the current part production status of the cell.
3760PRINT'"     ****Status of this cell ****"
3770PRINT'"Number of parts in input buffer  =";?in_partbuf
3780PRINT"Number of parts in output buffer =";?out_partbuf
3790PRINT"Number of processed parts        =";?nof_processed_parts
3800PRINT"Number of defective parts        =";?nof_defect_part
3810ENDPROC
>
```

L.

```
  10REM.............................. MONITOR ................................
  20REM This is a system and cell monitoring program which can also be used as a debugging tool
 to monitor the various status of the cells or system during a period of part production.
  30REM Assign a name to each state of memory locations within the CAMP.
  40warehouse%=20:ready=&FF:not_ready=&00:nothing=&09:busy=&FF:free=&00:connect=&FF:disconnect=
&00:processed=&00:processing=&FF:not_recognised=&01:run_prog=&FF:running=&FF:stop=&00:download_s
ucc=&01:transfer_part=&FF:transfer_part_comp=&00
  50completed=&00:idle=&01:stopped=&02:critical_op_in_progress=&03:device_reset=&04:device_inop
erable=&05:uncorrectable_error_det=&06:correctable_error_det=&07:diagnostic_running=&08
  60from_trolley=&01:to_trolley=&02:full=2
  70REM Flush all internal buffers.
  80*FX 15,0
  90OSWORD=&FFF1
 100OSBYTE=&FFF4
 110max_peek_length%=20
 120REM Reserve an area of memory for the control block.
 130DIM cblock%17,peekbuffer%max_peek_length%
 140CLS
 150PROCinput
 160PROCaddress
 170REPEAT
 180REM Get a snapshot of the CAMP in a distant entity.
 190peek_result%=FNpeek(st%,loc%,len%)
 200PROCresults
 210UNTIL FALSE
 220REM Check for any error.
 230ON peek_result%-&3F GOTO 240,260,280,300,320
 240PRINT"line jammed"
 250END
 260PRINT"net error"
 270END
 280PRINT"not listening"
 290END
 300PRINT"no clock"
 310END
 320PRINT"bad control block"
 330END
 340DEF PROCinput
 350INPUT"Which station...";st%
 360REM Set the base address.
 370loc%=31488
 380len%=20
 390ENDPROC
 400DEF FNpeek(station%,location%,length%)
 410LOCAL X%,Y%,A%
 420?cblock%=&81
 430cblock%?1=00
 440cblock%!2=station%
 450cblock%!4=peekbuffer%
 460cblock%!8=peekbuffer%+length%
 470cblock%!12=location%
 480X%=cblock%
 490Y%=cblock% DIV 256
 500A%=&10
 510CALL OSWORD
 520peekresult%=(U% AND &FF00) DIV 256
 530=peekresult%
 540DEF PROCresults
 550LOCAL J
 560FOR J=0 TO len%-1
```

```
570test_res$=FNtest_res
580PRINT TAB(23,J+1) test_res$
590NEXT J
600ENDPROC
610
620
630DEF PROCaddress
640REM Print the names of memory locations within the CAMP.
650CLS
660PRINT TAB(1,0)
670PRINT "7B00 ";"logical_assoc-----"
680PRINT "7B01 ";"client_st_no------"
690PRINT "7B02 ";"client_port-------"
700PRINT "7B03 ";"server_resp_mes---"
710PRINT "7B04 ";"message_status----"
720PRINT "7B05 ";"program_status----"
730PRINT "7B06 ";"input_partbuffer--"
740PRINT "7B07 ";"output_partbuffer-"
750PRINT "7B08 ";"production_period-"
760PRINT "7B09 ";"upload_prepration-"
770PRINT "7B0A ";"part_transfer-----"
780PRINT "7B0B ";"where_part_trans--"
790PRINT "7B0C ";"server_resp_part--"
800PRINT "7B0D ";"robot_status------"
810PRINT "7B0E ";"trolley_status----"
820PRINT "7B0F ";"device_status-----"
830PRINT "7B10 ";"defect_part_stat--"
840PRINT "7B11 ";"nof_defect_parts--"
850PRINT "7B12 ";"total_nof_parts---"
860PRINT "7B13 ";"no_processed_part-"
870PRINT''"   *** Monitoring station ";st%;" ***"
880ENDPROC
890
900
910DEF FNtest_res
920REM Test the state of each memory location within the CAMP before encoding it in the form o
f a message. This routine can be extended to accomodate the monitoring of any number of states w
ithin the CAMPs.
 930IF J=0 THEN IF peekbuffer%?J<>9 AND peekbuffer%?J=connect AND peekbuffer%?J=disconnect THEN
 GOTO 190
 940IF J=0 THEN IF peekbuffer%?J=9 THEN ="                 "
 950IF J=0 THEN IF peekbuffer%?J=connect THEN ="connect"+"          "
 960IF J=0 THEN IF peekbuffer%?J=disconnect THEN ="disconnect"+"      "
 970IF J=1 THEN IF peekbuffer%?J=9 THEN ="                 "
 980IF J=1 THEN IF peekbuffer%?J<>9 THEN =STR$(peekbuffer%?J)+"       "
 990IF J=2 THEN IF peekbuffer%?J<>9 THEN =STR$(peekbuffer%?J)+"       "
1000IF J=2 THEN IF peekbuffer%?J=9 THEN ="                "
1010IF J=3 THEN IF peekbuffer%?J=9 THEN  ="                "
1020IF J=3 THEN IF peekbuffer%?J=not_ready THEN ="not_ready"+"       "
1030IF J=3 THEN IF peekbuffer%?J=ready THEN ="ready"+"           "
1040IF J=4 THEN IF peekbuffer%?J=processing THEN ="processing"+"      "
1050IF J=4 THEN IF peekbuffer%?J=processed THEN ="processed"+"       "
1060IF J=4 THEN IF peekbuffer%?J=9 THEN ="                "
1070IF J=5 THEN IF peekbuffer%?J=9 THEN ="                "
1080IF J=5 THEN IF peekbuffer%?J=run_prog THEN="running"+"          "
1090IF J=5 THEN IF peekbuffer%?J=0 THEN="completed"
1100IF J=5 THEN IF peekbuffer%?J=stopped THEN="stopped"+"          "
1110IF J=6 THEN IF peekbuffer%?J=nothing AND st%=1 THEN ="               "
1120IF J=6 THEN IF peekbuffer%?J<>0 AND peekbuffer%?J<>full THEN =STR$(peekbuffer%?J)+"
      "

1130IF J=6 THEN IF peekbuffer%?J=0 THEN ="empty"+"             "
```

```
1140IF J=6 THEN IF peekbuffer%?J=full THEN ="full"+"          "
1150IF J=6 THEN IF peekbuffer%?J=9 THEN ="              "
1160IF J=7 THEN IF peekbuffer%?J=nothing AND st%=1 THEN ="              "
1170IF J=7 THEN IF peekbuffer%?J<>0 ANDpeekbuffer%?J<>full THEN =STR$(peekbuffer%?J)+"
"
1180IF J=7 THEN IF peekbuffer%?J=full THEN ="full"+"          "
1190IF J=7 THEN IF peekbuffer%?J=0 THEN ="empty"+"       "
1200IF J=8 THEN IF peekbuffer%?J=9 THEN ="              "
1210IF J=8 THEN IF peekbuffer%?J=&FF THEN ="started"+"    "
1220IF J=8 THEN IF peekbuffer%?J=0 THEN ="completed"
1230IF J=9 THEN IF peekbuffer%?J=9 THEN ="              "
1240IF J=9 THEN IF peekbuffer%?J=0 THEN ="completed"+"    "
1250IF J=9 THEN IF peekbuffer%?J=&FF THEN ="In_progress"+"    "
1260IF J=10 THEN IF peekbuffer%?J=9 THEN ="              "
1270IF J=10 THEN IF peekbuffer%?J=transfer_part_comp THEN ="complete"+"       "
1280IF J=10 THEN IF peekbuffer%?J=transfer_part THEN ="transfer"+"        "
1290IF J=11 THEN IF peekbuffer%?J=9 THEN ="              "
1300IF J=11 THEN IF peekbuffer%?J=0 THEN ="              "
1310IF J=11 THEN IF peekbuffer%?J=from_trolley THEN="from_trolley"+"   "
1320IF J=11 THEN IF peekbuffer%?J=to_trolley THEN="to_trolley"+"     "
1330IF J=12 THEN IF peekbuffer%?J=9 THEN ="              "
1340IF J=12 THEN IF peekbuffer%?J=ready THEN ="ready"+"          "
1350IF J=12 THEN IF peekbuffer%?J=not_ready THEN ="not_ready"+"      "
1360IF J=13 THEN IF peekbuffer%?J=9 THEN ="              "
1370IF J=13 THEN IF peekbuffer%?J=busy THEN ="busy"+"           "
1380IF J=13 THEN IF peekbuffer%?J=free THEN ="free"+"           "
1390IF J=14 THEN IF peekbuffer%?J=9 THEN ="              "
1400IF J=14 THEN IF peekbuffer%?J=busy THEN ="busy"+"           "
1410IF J=14 THEN IF peekbuffer%?J=free THEN ="free"+"           "
1420IF J=15 THEN IF peekbuffer%?J=9 THEN ="                 "
1430IF J=15 THEN IF peekbuffer%?J=0 THEN ="completed"+"      "
1440IF J=15 THEN IF peekbuffer%?J=1 THEN ="idle"+"           "
1450IF J=15 THEN IF peekbuffer%?J=2 THEN ="stopped"+"        "
1460IF J=15 THEN IF peekbuffer%?J=3 THEN ="critical_opration"
1470IF J=15 THEN IF peekbuffer%?J=4 THEN ="devise_is_reset"+" "
1480IF J=15 THEN IF peekbuffer%?J=5 THEN ="device_inoprable"
1490IF J=15 THEN IF peekbuffer%?J=6 THEN ="error_detected"+" "
1500IF J=15 THEN IF peekbuffer%?J=7 THEN ="error_detected"+" "
1510IF J=15 THEN IF peekbuffer%?J=8 THEN ="dignostic_run"+"  "
1520IF J=16 THEN IF peekbuffer%?J=9 THEN ="                 "
1530IF J=16 THEN IF peekbuffer%?J=&FF THEN ="defective_comp"
1540IF J=16 THEN IF peekbuffer%?J=0 THEN ="                "
1550IF J=17 THEN IF peekbuffer%?J=9 THEN ="                "
1560IF J=17 THEN IF peekbuffer%?J=0 THEN ="none"+"           "
1570IF J=17 THEN IF peekbuffer%?J<>0 AND peekbuffer%?J<>9 THEN =STR$(peekbuffer%?J)+"
"
1580REMIF J=18 THEN IF st%=1 THEN ="             "
1590IF J=18 THEN IF peekbuffer%?J=nothing AND st%=1 THEN ="             "
1600IF J=18 THEN IF peekbuffer%?J>=0 THEN =STR$(peekbuffer%?J)+"          "
1610IF J=19 THEN IF peekbuffer%?J>=0 THEN =STR$(peekbuffer%?J)+"          "
>
```

L.

```
   10REM........................ SCHEDULER ...................................
   20REM This program coordinates the overall activities of the FMS cells.
   30MODE 7:OSWORD=&FFF1:OSBYTE=&FFF4:*FX 15,0
   40REM Reset the system clock.
   50HIMEM=&7B00:M=&7B00:logi_associ=M+0:client_st_no=M+1:client_port=M+2:server_resp_mes=M+3:me
ss_stat=M+4:prog_stat=M+5:in_partbuf=M+6:out_partbuf=M+7:production_period=M+8:upload_prep=M+9:p
art_trans=M+10:where_part_trans=M+11
   60server_resp_part=M+12:robot_stat=M+13:trolley_stat=M+14:device_stat=M+15:defect_part_stat=M
+16:nof_defect_part=M+17:total_nof_parts=M+18:nof_processed_parts=M+19
   70warehouse%=20:ready=&FF:not_ready=&00:nothing=&09:busy=&FF:free=&00:connect=&FF:disconnect=
&00:processed=&00:processing=&FF:not_recognised=&01:run_prog=&FF:running=&FF:stop=&00:download_s
ucc=&01:transfer_part=&FF:transfer_part_comp=&00
   80completed=&00:not_compelete=&FF:idle=&01:stopped=&02:critical_op_in_progress=&03:device_res
et=&04:device_inoperable=&05:uncorrectable_error_det=&06:correctable_error_det=&07:diagnostic_ru
nning=&08
   90defect=&FF:started=&FF:in_progress=&FF:not_started=nothing
  100from_trolley=&01:to_trolley=&02:full=2:REM The size of part buffer is set to 5.
  110op_TYPE=1:key1=op_TYPE:CC_NAME=2:key2=CC_NAME:CC_CON=3:key3=CC_CON:STAT_NO=4:key4=STAT_NO:N
OF_PP=5:key5=NOF_PP:PP_NAME=6:key6=PP_NAME:SEQ_NO=7:key7=SEQ_NO:REM If the order of fields for a
 job definition is changed, then adjust the keys
  120REM Initialise the content of the dynamic database.
  130FOR I=0 TO 30:M?I=nothing:NEXT:?nof_defect_part=0:?nof_processed_parts=0:?total_nof_parts=0

  140CLS:max_recs=6
  150CLS:PRINT TAB(5,11):INPUT"Enter the scheduler station number :"sche_no%
  160rec_fields=12
  170max_peek_length%=5:max_poke_length%=5:current_rec=1
  180DIM cblock%40,peekbuffer% max_peek_length% ,pokebuffer% max_poke_length%
  190DIM field_cont$(rec_fields),index$(max_recs,2),field$(rec_fields),type$(rec_fields),width$(
rec_fields),dp$(rec_fields),command_stack$(8)
  200quit=FALSE
  210INPUT'"Enter the name of a predefined job for processing commencement :"job_name$:REPEAT:IN
PUT'"Enter the number of parts to be manufactured :"nof_parts%:UNTIL nof_parts%>0 AND nof_parts%
<=99
  220INPUT'"Enter the test number :"testno$
  230nof_parts_tr_from_w%=nof_parts%:nof_parts_tr_to_w%=nof_parts%:is_part_defective%=0
  240command_stack$(2)=job_name$:command_stack$(3)="INDEX":command_stack$(4)=job_name$+"_IND":PR
OCopen_file
  250REM If the ESCAPE key is pressed then terminate part production.
  260ON ERROR PROCterminate_part_prod
  270FOR I=1 TO VAL(nofrec$)
  280REM Intialise the cell controllers before sending the name of a job to be processed.
  290IF LEFT$(FNget_field(VAL(index$(I,2)),key3),2)="IN" THEN GOTO 330
  300s%=FNget_field(VAL(index$(I,2)),key4):mes$="*I AM SAEID":PROC_keyboard_poke(s%,mes$):mes$="
CH."+""""+FNget_field(VAL(index$(I,2)),key2)+"""":PROC_keyboard_poke(s%,mes$)
  310PRINT TAB(0,23) STRING$(20," "):PRINT TAB(1,23) "Wait...":loc%=prog_stat:REPEAT:PROCpeek(s%
,loc%,1):UNTIL peekbuffer%?0=nothing:PROCdelay(1200):pokebuffer%?0=run_prog:PROCpoke(s%,loc%,1):
mes$="JOB NAME "+job_name$
  320PROCtx_message(sche_no%,s%,mes$):PROCdelay(100):mes$="NOF PARTS"+STR$(nof_parts%):PROCtx_me
ssage(sche_no%,s%,mes$)
  330NEXT I
  340command_stack$(2)=job_name$:PROCclose_file
  350command_stack$(2)=job_name$:command_stack$(3)="INDEX":command_stack$(4)=job_name$+"_INS":PR
OCopen_file
  360PROCseq_active_cell_check
  370FOR I=1 TO VAL(nofrec$)
  380IF index$(I,1)="00" THEN NEXT I
  390restof_recs%=I
  400starting_recs%=VAL(index$(I,2)):starting_st%=FNget_field(starting_recs%,key4)
  410job_timing$=job_name$+"_A"+testno$
  420act_ch=OPENOUT(job_timing$)
  430PRINT£act_ch,STR$(nof_parts%)
  440@%=&01020209
  450REPEAT
  460loc%=in_partbuf:s%=starting_st%:PROCpeek(s%,loc%,1)
  470IF peekbuffer%?0<>full THEN PROCpart_trans_from_warehouse
  480PROCoutput_partbuf_check
  490UNTIL nof_parts_tr_from_w%=0
  500REPEAT
  510REM Check whether a part is needed to be transferred to or from a cell.
  520PROCoutput_partbuf_check
  530IF is_part_defective%=defect OR nof_parts_tr_from_w%<>0 THEN GOTO 450
  540UNTIL nof_parts_tr_to_w%=0
  550CLOSE£act_ch
  560PROCproduction_period_comp
  570command_stack$(2)=job_name$:PROCclose_file
  580PRINT TAB(0,23) STRING$(30," "):PRINT TAB(1,23) "Wait..."
  590@%=10
  600D%=8:CHAIN"MASTER"
  610END
  620
  630
  640REM Include the manufacturing database here.
  650REM The communication routine resids here.
  660
  670
  680DEF PROCseq_active_cell_check
  690REM Check whether all the particiant cells for current job are active.
  700LOCAL I
  710FOR I=1 TO VAL(nofrec$)
  720IF index$(I,1)="00" THEN IF LEFT$(FNget_field(VAL(index$(I,2)),key3),1)="A" THEN PRINT "INV
ALID ACTIVE CELL IN RECORD ";VAL(index$(I,2)):ENDPROC
  730IF index$(I,1)<>"00" THEN IF LEFT$(FNget_field(VAL(index$(I,2)),key3),1)="I" THEN PRINT "IN
VALID SEQUENCE IN RECORD =";VAL(index$(I,2)):ENDPROC
  740NEXT I
  750ENDPROC
  760
  770
  780DEF PROCpart_trans_from_warehouse
  790PRINT"--Part transfer from warehouse in progress--"
  800s%=warehouse%:fromto_trolley$="to_trolley"
  810cell_val=0
  820act_mes$=STR$(cell_val)+"R"+STR$(TIME/6000):PRINT£act_ch,act_mes$
  830PROCsend_trolley(s%,fromto_trolley$)
  840nof_parts_tr_from_w%=nof_parts_tr_from_w%-1:s%=starting_st%:fromto_trolley$="from_trolley":
PROCsend_trolley(s%,fromto_trolley$)
  850PRINT"Part transfered to cell (ST.";s%;")"
  860cell_val=1
  870act_mes$=STR$(cell_val)+"T"+STR$(TIME/6000):PRINT£act_ch,act_mes$
  880ENDPROC
  890
  900
  910DEF PROCsend_trolley(st%,fromto_trolley$)
  920REM The routine to control the physical movement of the trolley to be included here.
  930?trolley_stat=busy
  940IF st%=warehouse% THEN PROCdummy_st:?trolley_stat=free:ENDPROC
  950pokebuffer%?0=transfer_part:loc%=part_trans:PROCpoke(st%,loc%,1)
  960PROCdrive_trolley(st%)
  970IF fromto_trolley$="from_trolley" THEN pokebuffer%?0=from_trolley:loc%=where_part_trans:PRO
Cpoke(st%,loc%,1)
  980IF fromto_trolley$="to_trolley" THEN pokebuffer%?0=to_trolley:loc%=where_part_trans:PROCpok
```

```
e(st%,loc%,1)
  990REPEAT:loc%=server_resp_part:PROCpeek(st%,loc%,1):UNTIL peekbuffer%?0=ready
 1000REPEAT:loc%=part_trans:PROCpeek(st%,loc%,1):UNTIL peekbuffer%?0=transfer_part_comp
 1010?trolley_stat=free
 1020ENDPROC
 1030
 1040
 1050DEF PROCdrive_trolley(st%)
 1060PRINT"--Driving the trolly to cell (ST.";st%;")--"
 1070PRINT"--Trolley is in cell (ST.";st%;")--"
 1080
 1090ENDPROC
 1100
 1110
 1120DEF PROCdummy_st
 1130REM The logical address of the warehouse to be included here.
 1140PRINT"--Trolley is in the warehouse now--"
 1150ENDPROC
 1160
 1170
 1180DEF PROCoutput_partbuf_check
 1190REM Check the status of the output part buffer of all the active cells.
 1200LOCAL J
 1210pointer%=0:is_part_defective%=nothing
 1220FOR J=restof_recs% TO VAL(nofrec$)
 1230rec=VAL(index$(J,2)):s%=FNget_field(rec,key4):loc%=out_partbuf
 1240PROCpeek(s%,loc%,1)
 1250IF peekbuffer%?0<>0 THEN pointer%=J:loc%=defect_part_stat:PROCpeek(s%,loc%,1):is_part_defec
tive%=peekbuffer%?0:PROCinput_partbuf_check
 1260NEXT J
 1270ENDPROC
 1280
 1290
 1300DEF PROCinput_partbuf_check
 1310REM Before sending a part to a cell check the status of the input part buffer for that cell
.
 1320IF pointer%=VAL(nofrec$) AND is_part_defective%<>defect THEN PROCpart_trans_to_warehouse:EN
DPROC
 1330IF is_part_defective%=defect THEN PROCdefective_part_trans_to_warehouse:ENDPROC
 1340rec=VAL(index$(pointer%+1,2)):s%=FNget_field(rec,key4):loc%=in_partbuf:PROCpeek(s%,loc%,1)
 1350IF peekbuffer%?0=full THEN ENDPROC
 1360PROCpart_trans_to_nextcell
 1370ENDPROC
 1380
 1390
 1400DEF PROCpart_trans_to_nextcell
 1410REM Transfer a part from one cell to another.
 1420rec=VAL(index$(pointer%,2)):s%=FNget_field(rec,key4):fromto_trolley$="to_trolley":PROCsend_
trolley(s%,fromto_trolley$)
 1430PRINT"Part transfer from cell (ST.";s%;")"
 1440act_mes$=STR$(VAL(index$(pointer%,1)))+"F"+STR$(TIME/6000):PRINT£act_ch,act_mes$
 1450rec=VAL(index$(pointer%+1,2)):s%=FNget_field(rec,key4):fromto_trolley$="from_trolley":PROCs
end_trolley(s%,fromto_trolley$)
 1460PRINT"Part transfered to cell (ST.";s%;")"
 1470act_mes$=STR$(VAL(index$(pointer%+1,1)))+"T"+STR$(TIME/6000):PRINT£act_ch,act_mes$
 1480ENDPROC
 1490
 1500
 1510DEF PROCpart_trans_to_warehouse
 1520REM Transfer a good part to the warehouse.
 1530rec=VAL(index$(pointer%,2)):s%=FNget_field(rec,key4):fromto_trolley$="to_trolley":PROCsend_
trolley(s%,fromto_trolley$)
 1540act_mes$=STR$(VAL(index$(pointer%,1)))+"F"+STR$(TIME/6000):PRINT£act_ch,act_mes$
 1550PRINT"--Part transfer to warehouse in progress--"
 1560PRINT"Part transfer from cell (ST.";s%;")"
 1570s%=warehouse%:fromto_trolley$="from_trolley":nof_parts_tr_to_w%=nof_parts_tr_to_w%-1:PROCse
nd_trolley(s%,fromto_trolley$)
 1580cell_val=0
 1590act_mes$=STR$(cell_val)+"G"+STR$(TIME/6000):PRINT£act_ch,act_mes$
 1600b=?nof_processed_parts:b=b+1:?nof_processed_parts=b:b=?total_nof_parts:b=b+1:?total_nof_par
ts=b
 1610ENDPROC
 1620
 1630
 1640DEF PROCdefective_part_trans_to_warehouse
 1650REM Transfer a defective part to the defective part store section of the warehouse.
 1660rec=VAL(index$(pointer%,2)):s%=FNget_field(rec,key4):fromto_trolley$="to_trolley":PROCsend_
trolley(s%,fromto_trolley$)
 1670act_mes$=STR$(VAL(index$(pointer%,1)))+"F"+STR$(TIME/6000):PRINT£act_ch,act_mes$
 1680loc%=defect_part_stat:pokebuffer%?0=nothing:PROCpoke(s%,loc%,1)
 1690PRINT"!!!Defective part transfer to warehouse!!!"
 1700s%=warehouse%:fromto_trolley$="from_trolley":nof_parts_tr_from_w%=nof_parts_tr_from_w%+1:PR
OCsend_trolley(s%,fromto_trolley$):b=?nof_defect_part:b=b+1:?nof_defect_part=b
 1710cell_val=0:act_mes$=STR$(cell_val)+"D"+STR$(TIME/6000):PRINT£act_ch,act_mes$
 1720b=?total_nof_parts:b=b+1:?total_nof_parts=b
 1730ENDPROC
 1740
 1750
 1760DEF PROCproduction_period_comp
 1770REM Send command to all active cell that the end of a production period should be commenced
.
 1780FOR I=1 TO VAL(nofrec$)
 1790IF index$(I,1)<>"00" THEN s%=FNget_field(VAL(index$(I,2)),key4):loc%=production_period:poke
buffer%?0=completed:PROCpoke(s%,loc%,1)
 1800NEXT I
 1810ENDPROC
 1820
 1830DEF PROCterminate_part_prod
 1840REM Send commands to all active cell to end the part production in their cells.
 1850LOCAL I
 1860FOR I=restof_recs% TO VAL(nofrec$)
 1870rec=VAL(index$(I,2)):s%=FNget_field(rec,key4):loc%=prog_stat
 1880pokebuffer%?0=stopped:PROCpoke(s%,loc%,1)
 1890NEXT I
 1900CLOSE£0
 1910STOP
 1920ENDPROC
>
```