Flexible Manufacturing System Software Development Using Simulation

by

Timothy Patrick Martin

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
Master of Science
in
Electrical Engineering

APPROVED:

Dr. Charles E. Nunnally,
Chairman

Dr. Joseph G. Tront

Dr. Hugh F. VanLandingham

June 1985

Blacksburg, Virginia

Flexible Manufacturing System Software Development Using Simulation

by

Timothy Patrick Martin

Dr. Charles E. Nunnally, Chairman

Electrical Engineering

(ABSTRACT)

This paper presents a hierarchical modeling method that can be used to simulate a Flexible Manufacturing System (FMS) at all levels of detail. The method was developed specifically to aid the software development needed for the hierarchy of computers that are present in an FMS. The method was developed by modeling an existing FMS. The models developed of the existing FMS are described in detail to provide an example of how to model other FMSs. The basic building blocks needed for designing other FMSs with this modeling method are provided.

The models were written in the SIMAN simulation language. SIMAN was found to be an easy language to use for the hierarchical modeling of FMSs.

ACKNOWLEDGEMENTS

I thank my advisor, Dr. Charles E. Nunnally, for his guidance and encouragement during the course of this research. I thank him especially for his willingness to go out of his way to provide the equipment and resources to make the research easier.

I thank Dr. Joseph G. Tront and Dr. Hugh F. VanLandingham for serving on my committee.

I thank Microcompatibles for the use of their Grafmatic and Plotmatic software, which aided greatly the development of the plotting program.

I extend special thanks to my wife, Kathy, for her encouragement and companionship throughout this project. I thank her especially for her endurance at the end.

I dedicate this work to Jesus Christ, from whom all knowledge comes. I pray that this work will glorify Him. I thank God for providing the wisdom needed to complete the research and for providing the endurance to get it done.

Finally, I thank E. I. du Pont de Nemours & Co. for patiently waiting for me to finish this thesis. I also thank them for providing some of the resources to allow me to finish the final details on this paper.

TABLE OF CONTENTS

1.0 Introduction 1

1.1 Background 1

1.2 Description of the Problem Solved 3

2.0 Overview of Research 6

2.1 Design Procedure 7

2.2 The SIMAN Simulation Language 8

2.3 IBM PC Usage 10

2.4 The Virginia Tech Miniature FMS 11

2.5 Flow Diagrams 17

2.6 High Level Model 18

2.7 High Level Plot 18

2.8 Detailed Model 19

3.0 High Level Model 21

3.1 Overview of the Model 21

3.2 Model Description 23

3.2.1 Pallet Arrival Block 25

3.2.2 Move to the Next Station Block 27

3.2.3 The Machine Station Block 30

3.2.4 The Exit Conveyor Block 32

3.3 The Experiment Frame 32

3.4 Standard SIMAN Output 35

3.5 The Trace 38
3.6 High Level Plot 41
3.7 Verification of the High Level Model 44
3.8 Modifying the Model 44
3.9 Sample Runs 45

4.0 The Detailed Model 50
4.1 Overview of the Model 50
4.2 Representing Computer Storage 54
4.3 The Modeling of Polling and Interrupt Routines . 54
4.4 Description of the Events Used 57
4.5 Description of the Model Blocks 58
4.5.1 The Pallet Arrival Block 58
4.5.2 The Task List Block 60
4.5.3 The System-Host Polling Routine Block 60
4.5.4 The Pallet Processing Block 64
4.5.5 The Machine Processing Block 66
4.5.6 The Exit Conveyor Block 68
4.5.7 The System-Host Interrupt Routine 68
4.5.8 The Mini-Host Model 71
4.5.9 The Axis-Controller Model 74
4.6 The Experimental Frame 75
4.7 Output Available 77
4.8 Initialization of the Simulation 78
4.9 Verification of the Model 79
4.10 Modifying the Model 79

4.11 Sample Run 80

5.0 Conclusions and Recommendations for Further Research 83

5.1 Conclusions 83

5.2 Suggestions for Further Research 85

References 87

Bibliography 90

Appendix A. Introduction to SIMAN 94

A.1 Introduction to Computer Simulation Modeling . . 94

A.1.1 Modeling and Simulation 94

A.1.2 Modeling Orientations 95

A.1.2.1 Continuous Modeling 96

A.1.2.2 Discrete Modeling 97

A.1.3 Computer Languages for Simulation 99

A.2 Overview of SIMAN 101

A.2.1 The SIMAN Framework 102

A.2.2 Essential Terms 104

A.2.3 Simulation Process Flow 107

A.2.4 SIMAN Variables 109

A.3 The Model Frame 113

A.4 The Experiment Frame 128

A.5 Output Available 139

A.5.1 Run Time Output 139

A.5.2 The Output Frame 140
A.6 The Event Frame 143
A.7 The Continuous Frame 146
A.8 Evaluation and Conclusion 148
A.8.1 Evaluation of SIMAN 149
A.8.2 Conclusion 152

Appendix B. Notes on Using FORTRAN 154
B.1 Incompatibility Between Different Versions of Microsoft FORTRAN 154
B.2 Compatibility Between Different Models of the IBM PC 155
B.3 Abnormal Exit from the Compiler or Linker . . . 155

Appendix C. IBM PC Usage 157
C.1 Disk Layout 157
C.2 Translating and linking SIMAN models 159
C.3 How to run the High Level and Detailed Models . 160
C.4 Running the Plot Program 162
C.5 Compiling FORTRAN 166
C.6 Linking FORTRAN Routines to SIMAN 167
C.7 Linking the Plot Program 168
C.8 Conclusion 170

Appendix D. High Level Model Listing 172
D.1 HL.DES 172

D.2 HL.MOD 176
D.3 HL.EXP 180

Appendix E. High Level Plot Listings 183
E.1 PLOT.FOR 183
E.2 INPUT.FOR 190
E.3 SCREEN.FOR 203
E.4 PLOTTER.FOR 209
E.5 MACH.FOR 215
E.6 ROBOT.FOR 220
E.7 MODELCOM.FOR 225
E.8 PLOTCOM.FOR 225

Appendix F. Detailed Model Listing 226
F.1 FMS.DES 226
F.2 FMS.MOD 238
F.3 FMS.EXP 252

Appendix G. FMS FORTRAN Listings 256
G.1 EVENT.FOR 256
G.2 PRIME.FOR 265
G.3 TRACE.FOR 270

Appendix H. SIMAN Preprocessor 284

Vita 291

LIST OF ILLUSTRATIONS

Figure 2.1 The VTMFMS Layout 13

Figure 2.2 The Computer Hierarchy of the VTMFMS . . 15

Figure 3.1 High Level Model Flow 22

Figure 3.2 Pallet Arrival Block 26

Figure 3.3 Move to the Next Station Block 28

Figure 3.4 The Machine Stations 31

Figure 3.5 The Exit Conveyor 33

Figure 3.6 SIMAN Standard Summary Report 36

Figure 3.7 Sample Trace Output 40

Figure 3.8 Sample Plot Output 43

Figure 3.9 Modified Machine Station Block to Include Storage for one Pallet 46

Figure 3.10 Modified Model Output 47

Figure 4.1 Detailed Model Blocks and Their Connections 52

Figure 4.2 The Pallet Arrival Block Flow 59

Figure 4.3 System Host Polling Routine Flow 62

Figure 4.4 The Pallet Processing Block Flow 65

Figure 4.5 The Machine Processing Block Flow 67

Figure 4.6 The Exit Conveyor Block Flow 69

Figure 4.7 The System-Host Interrupt Routine Flow . . 70

Figure 4.8 The Mini-Host Polling Routines Flow . . . 73

Figure 4.9 The Axis-Controller Flow 76

Figure 4.10 Sample Detailed Model Summary Report . . 81

Figure 4.11 Sample Detailed Model High Level Trace . 82

Figure A.1 SIMAN User Assignable Variables 110

Figure A.2 SIMAN System Variables 111

Figure A.3 SIMAN Random Variables 112

Figure A.4 SIMAN Block Modifiers 114

Figure A.5 SIMAN Block Types 116

Figure A.6 Operation Block Type Functions (Page 1 of 3) 117

Figure A.7 Operation Block Type Functions (Continued, Page 2 of 3) 118

Figure A.8 Operation Block Type Functions (Continued, Page 3 of 3) 119

Figure A.9 Transfer Block Type Functions 120

Figure A.10 Hold Block Type Functions (Page 1 of 2) 121

Figure A.11 Hold Block Type Functions (Continued, Page 2 of 2) 122

Figure A.12 SIMAN Overall Defining Experiment Elements 130

Figure A.13 SIMAN Parameter Definition Experiment Elements 131

Figure A.14 SIMAN Resource Definition Experiment Elements 132

Figure A.15 SIMAN Initialization Experiment Elements 133

Figure A.16 SIMAN Output Experiment Elements . . . 134

Figure A.17 SIMAN Output Frame Elements 141

Figure C.1 Sample PLOT Session 165

Chapter 1

INTRODUCTION

Over the last twenty five years, the state of the art in numerically controlled machine tools has advanced considerably. The state of the art has changed from a stand alone numerically controlled machine tool with paper tape input to an entire factory of digitally controlled tools and material handlers all linked by a hierarchy of computers. The increased complexity of the systems has generated a need for methods to design these systems efficiently. This thesis presents a method to ease the design of the hardware and software needed to implement one of these integrated manufacturing systems.

1.1 Background

In the 1950's, numerical control (NC) was introduced to manufacture parts with a precision that was higher than could easily be obtained manually [1], [2]. NC was soon used because it allowed higher production rates than manual machining, and it was less expensive to implement than fixed (assembly line) automation when a variety of parts was to be manufactured. NC machines processed a part by reading a manually loaded paper tape that holds the "part program" for that part. The controller used hybrid analog-digital

"hardwired" control techniques to drive the machine from the data on the tape. To generate the tape, an "English like" manuscript that describes in detail the exact path the cutting tool is to take is translated into the part program needed by the NC [3]. The computer then punches the tape. If any change is needed in the part program, a new paper tape has to be generated.

In the 1960's, Computer Numerical Control (CNC) was introduced [4], [5]. In CNC, a dedicated computer is used to replace the "hardwired" machine controller. By using a "softwired" controller, the cutting control algorithm can be much more complex and is much easier to change than with the old "hardwired" controller. Another benefit is that the CNC usually has enough memory so that often used part programs can be loaded into memory so that when ever they are needed, they can be used from memory without needing to re-read the paper tape. The CNC often allows the user to edit the part program so that a specialized part can be made without having to make a new paper tape.

Introduced at about the same time as CNC, Direct Numerical Control (DNC) replaces the paper tape reader with a communications line to a computer [1], [4], [6]. The part program is stored in the computer's memory. As the NC needs more data, the computer down loads the data to the NC. The computer can be used for many NC tools at the same time, reducing the cost of the system. An important function

allowed by DNC is that data can be sent from the NC to the computer to aid in the management of the machine shop. This allows the management of the parts movement and machine usage in the shop to become "closed-loop" [7].

By the 1970's, automatic computer controlled material handlers were added to manufacturing systems to form a Flexible Manufacturing System (FMS) [8], [9]. These systems allow the manufacture of a variety of parts by the same system without any manual reconfiguration of the system between part types. An FMS consists of CNC machines, CNC material handlers and a DNC supervisory computer. The CNCs and DNC form a hierarchy of computers. Additional levels of computer hierarchy are often provided by an up link to management and engineering computers. In an FMS the raw materials are queued at the input of the system and the finished parts are removed at the output of the system. All of the processing is computer controlled. The actual control of the machines and material handlers is provided by the software in the system. The software is also responsible for the scheduling of what part should move to what machine and what part program is to be used on that machine.

1.2 Description of the Problem Solved

An FMS is much more complex to design than a system of stand alone NC machines. For the NC machines, the main

design problem is to come up with a good mix of machines for the system. This is only the start of the design process for an FMS. An appropriate material handler must also be found. A control strategy for the movement of the parts in the system must be developed. A hierarchy of computers must be designed to control all of the elements of the system. Each of the computers must have its software designed and coded. Finally, the system must be built and tested.

Simulation is one of the most important tools used in designing an FMS. Simulation is usually used to find a suitable configuration for the system [10] - [13]. Such factors as the number of machines, what type of material handlers to use, the amount of storage needed at the stations and what control strategy should be used to move parts are determined using simulation.

Once the configuration has been designed, the software for the hierarchy of computers must be written. The function of the software has already been determined by the configuration simulation, but the algorithm to implement the function has not been written at this stage. Simulation is not commonly used for this part of the design, even though the software design can be quite complex. It is a very difficult task to coordinate the software from the different machines. Debugging is difficult, especially since computers from different manufacturers are frequently used for each part of the system [14], [15]. There is a need for software

development tools, so most of the software development time will not be spent debugging the software, as it now is [16]. During this phase other factors must be decided, such as the best communication technique and the division of tasks between the computers. What is needed is a way to easily simulate the hardware and software, so the software can be designed using the aid of simulation. By using simulation the software can be checked for correct operation and the system's ability to tolerate faults can be checked, even under the most stressful conditions. Alternate algorithms can be easily compared. The software complexity can be approximated by evaluating the model. This thesis presents a simulation method which can be used for both the configuration design and the software design of an FMS.

Chapter 2

OVERVIEW OF RESEARCH

The research presented in this thesis was conducted to find a methodology that can be used to aid in the design of software for a hierarchy of control computers. The method was developed specifically for Flexible Manufacturing Systems, but it can be applied to other types of systems. The method involves hierarchical modeling and simulation of the system being designed. Techniques for making initial designs are not provided, only a method for testing a given design is given.

The method was developed by simulating an existing FMS. A macro model was created first. A graphical output program was developed to present the results of the macro simulation. From the macro model a micro model was developed and simulated.

An existing FMS was used in the research to guarantee that the models developed represented a realizable system. Verification of the simulation model was much easier than it would have been if the physical system did not exist since the simulation results could be checked against the actual system performance. Modeling an already existing system sped up the research because the system did not have to be designed as the model was created. An additional benefit of simulating a system that exists is that the ease of

simulating a system that does not fit into the "mold" created by the simulation language can be checked.

Even though an existing FMS was used, the methodology can still be used to design a new FMS. The designer can use the models developed in this research as examples. The methodology presented here gives the building blocks needed to develop models of an FMS system.

2.1 Design Procedure

The first part of the design procedure is to develop the requirements of the system. Once they have been determined, a configuration of machines and material handlers must be determined. This is done with the first level of modeling. When the best configuration is determined, more detailed models are developed to define what computer hardware is needed and to develop the software algorithms. Once the entire system is working on the simulator, the hardware can be purchased and the software written. The final steps are putting everything together, and then testing and debugging the physical system. The final step should be much easier using this method since the system was shown to work in the simulation [17].

The method uses a hierarchy of models. By using a hierarchy, the less detailed models can be used to debug and verify the more detailed models. The structure of the

detailed models will be almost the same as that of the less detailed models. This allows the same output to be gathered from each of the models. The user's understanding of the model is aided by the hierarchical approach since the model is broken into small, understandable pieces that directly relate to a part of the physical system. Other advantages to the hierarchical approach are that results are obtained in the order they are needed and that the development costs are lower than with a non-hierarchical approach [18].

2.2 The SIMAN Simulation Language

The models were written in the general purpose simulation language SIMAN. A simulation language was used because the special features of the language simplify the simulation procedure considerably. Models written in SIMAN are also much easier to understand than models written in a general purpose language, such as FORTRAN.

SIMAN is a relatively new general purpose simulation language [19]. It is described in detail in the appendix "Introduction to SIMAN" on page 94. SIMAN is a FORTRAN based, combined discrete-continuous simulation language. The main emphasis of SIMAN is discrete process orientation modeling. Event orientation and continuous models are used by linking FORTRAN subroutines to SIMAN. SIMAN runs on both mainframes

and personal computers (PCs). The two versions are compatible.

SIMAN was chosen for the advantages it has over other simulation languages. SIMAN is a modular language, which suits the problem to be solved. The same set of code can represent many different machines, which is ideal for the modeling of the repeated machines or controllers which are common in FMSs. The model is separate from the "experiment". The experiment contains all of the parameters of the system. This allows many cases to be run without having to modify the model at all. SIMAN has more advanced functions than its predecessors. The advanced features that are most pertinent to this research are the material handling functions. These make it very easy to model robots and conveyors.

SIMAN allows the modeler to write additional model functions. These are written in FORTRAN, usually with calls to SIMAN routines. This allows complicated functions to be "hidden" from the model - only the call to the new function is evident. SIMAN also allows continuous models to be added to the discrete models. This will enable the designer to check motor control algorithms for correct operation using the same framework that is used for the rest of the models. (No continuous models were written for this research).

SIMAN has several problems that make it less than ideal to work with. SIMAN does not allow comments to be placed on a statement that is continued on the next line. This makes

it difficult to comment long and complicated statements. This problem is especially acute in the experiment frame where the statements are very data intensive and are very long. The author fixed this problem by writing a pre-processor that strips off all of the comments in the source file. A comment is anything after a semicolon (the SIMAN statement terminator) or a tilde (a comment symbol specified for use with the pre-processor, it shows up as "¬" on this printer). The pre-processor is called SIMPRE and works with both the experiment and model files. The program listing is in the appendix "SIMAN Preprocessor" on page 284. Passing parameters to FORTRAN events from the model frame can sometimes be cumbersome. This reduces the advantage of using the events for clarity if many parameters need to be passed.

Overall, SIMAN is the best language the author has seen for writing models of FMSs.

2.3 IBM PC Usage

I did all of the computer work for this research on the IBM PC family of machines. Working on a PC can sometimes be an advantage. One of the main advantages is that you don't have to worry about how much "computer money" you are spending, you can use the PC as much as you are willing to sit there. Also, programs are very transportable. IBM PCs are much easier to find than a mainframe with the software

you need. Portability is convenient when the machine breaks, because you can just move to another PC. The PC tends to give a "friendlier" feeling and the attitude that you are in control since it is right there in front of you, not locked away someplace. The disadvantages of using the PC are that it is sometimes quite slow and the diskettes have limited storage, requiring a box or two to store all of the pertinent data and programs.

Another problem with using the PC has to do with FORTRAN. The object code from different versions of Microsoft FORTRAN is **not** compatible with any other version of Microsoft FORTRAN. This and other aspects of using FORTRAN on an IBM PC is discussed in the appendix "Notes on Using FORTRAN" on page 154.

The disk setup used on the IBM PC and how to run all of the programs developed is described in the appendix "IBM PC Usage" on page 157.

2.4 The Virginia Tech Miniature FMS

The Virginia Tech Miniature Flexible Manufacturing System (VTMFMS) was used as the sample system in developing the modeling methods. The system was developed at Virginia Tech as a joint project by the Industrial Engineering and Electrical Engineering departments. The development of the VTMFMS is described in references [20] through [27].

The VTMFMS consists of four machining stations, an input conveyor, an exit conveyor and a programmable part mover (robot). The four machining stations and two conveyors are spaced almost equidistant in a circle around the robot. The system layout is shown in Figure 2.1 on page 13. Only two of the machines have been implemented. One is a three-axis and the other is a four-axis milling machine. They are each approximately 12" x 12" x 18" [28]. The robot is a three axis machine that can move a four inch square pallet. Currently the conveyors have no storage, they are just pick-up and place points. There is no storage at the machining stations. Even though the system is not physically complete, it has been modeled as a complete system.

There are three levels of computers in the system. An additional level of computers can be connected to the system, but this level is not used to control the real-time operation of the FMS, it is only used for developing part programs which are then down loaded to the FMS. The real-time operation of the FMS is the only concern of this research, so only the lower three levels of computers are modeled. The highest level computer modeled must reserve some capacity to take care of any (non-modeled) communications with the higher level computers.

The computers in the system and their connections to other parts of the system are shown in Figure 2.2 on page 15. The system-host is responsible for the overall control

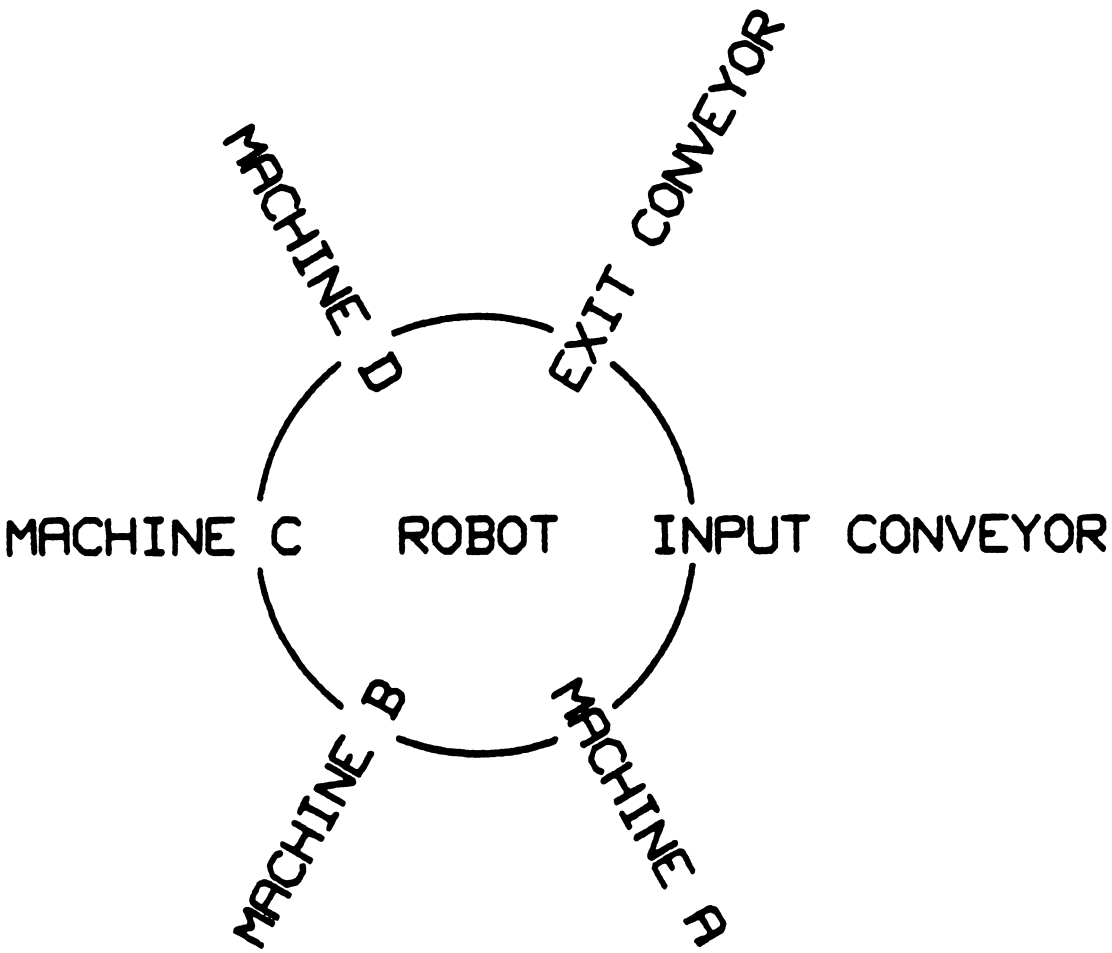

Figure 2.1 The VTMFMS Layout

of the FMS. It is told what part is to be manufactured and it is then responsible for manufacturing the part. The mini-hosts are responsible for overseeing a machine. They receive part programs from the system host and send commands to the axis-controllers. The axis-controllers control the motors.

Most of the communication in the system is done over asynchronous serial lines. The system-host communicates totally in this form, using an ASCII data format and software handshaking.[1] The mini-host - axis-controller communication is in binary form over asynchronous serial lines with hardware handshaking. There are also some dedicated signal lines between them (in addition to the hardware handshaking lines). The communication between the axis-controller and machine is all on dedicated signal lines. The communications software is both polling and interrupt driven.

The **system-host** is an LSI-11 mini-computer with two floppy disk drives. Most of the system's software is for the system-host. Managing the system involves many things. The

---

[1] Software handshaking is done over the same line as the data is transmitted. The handshaking signals can be considered part of the data character set. Hardware handshaking is done with additional lines between the transmitter and receiver. The handshaking signals are transparent to the software. No data space is taken. Dedicated signal lines are used to indicate the state of a single condition or command at any instant in time. The line is dedicated to one condition and can not be used for other communication.

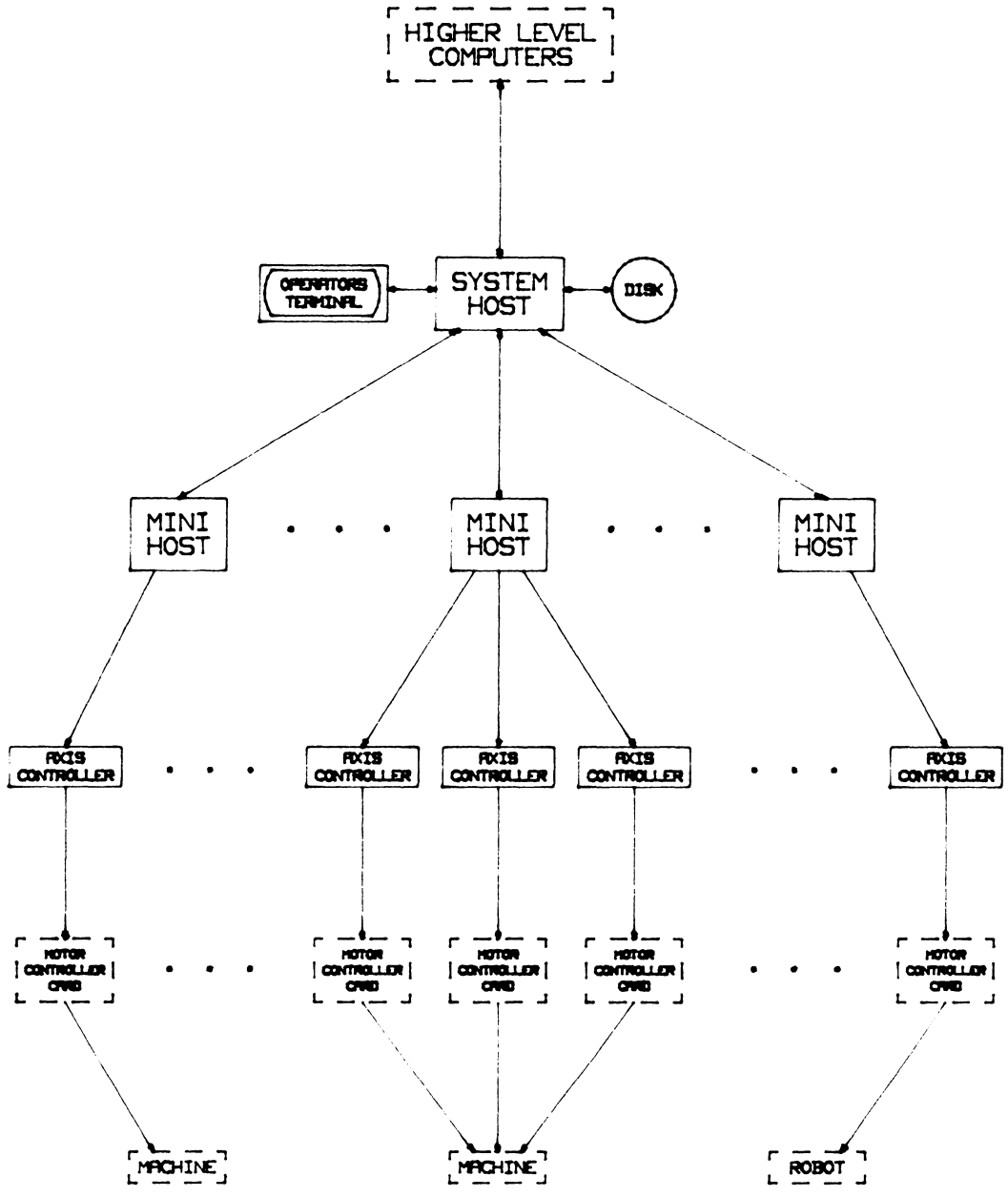

Figure 2.2 The Computer Hierarchy of the VTMFMS

system-host must decide the strategy to manufacture the parts it is told to manufacture. It decides what machine the part is to go to and what parts have priority. It distributes the part programs, as needed, to the mini-hosts. It generates the commands to move the robot. It also detects errors, provides diagnostics and responds appropriately to the situation. The operator and higher level computer interfaces are provided by the system-host.

The **mini-host** is responsible for supervising the operation of a machine, including the axis-controllers. The mini-host requests and receives part programs from the system host. It converts the programs into binary form and sends the appropriate data to each of the axis-controllers. It coordinates the actions of the axis-controllers. The mini-host must check the status of the machines for errors and report any errors to the system-host. The mini-host is responsible for bringing the machine to its "home" position, which is where the robot can pick up or drop off a pallet. The mini-host is an Intel 8751 single chip microcomputer.

The **axis-controllers** are also Intel 8751s. The axis-controller is responsible for the operation of one axis of the machine. It receives commands in binary form. It outputs the signals necessary to drive the motor driver card. It also checks for an end of travel on its axis. The present VTMFMS uses only stepper motors running open loop. If the

loop were closed, the control algorithm would be placed in the axis-controller.

As the software is set up now, all machines are considered to be different and can not be substituted for any other machine. A pallet can only move to a machine that is in increasing alphabetical order from the machine it is on. This is to prevent a deadlock from occurring. If, for example, the pallet at machine A needs to go to machine B and the pallet at machine B needs to go to machine A, a lock up will occur since there is no storage space in the system. The machine sequence for any pallet type is fixed. The processing time for a pallet type on a particular machine is fixed. There may be any number of pallet types.

2.5 Flow Diagrams

When I originally modeled the VTMFMS I modeled it in flow diagram form. This is similar to a flow chart, except that time delays are a major part of the diagram. Things that occur in an instant of time are placed in a box. Logical branches are indicated by thin lines and include the condition needed to transverse the branch. Time delays are indicated by thick lines and include the length of the delay or what condition ends the delay. Examples of the flow diagrams are in the following chapters.

2.6 High Level Model

The first step in the modeling of the system was creating the high level model. This model would be used to find a good configuration for the system. The model is fairly simple and can be verified rather easily. The machines, material handler and conveyors were modeled without modeling the computers. The control strategy for choosing which machine is the most complicated part of the model. The main reasons for developing this model are to verify that the machine mix chosen is sufficient, that the material handler is adequate and to find the best control strategy for moving parts. An additional reason for developing this model is that it is a stepping stone to the more detailed models. The values that are to be optimized are system through-put and machine utilization. The high level model developed is described in the chapter "High Level Model" on page 21 and the model is printed in the appendix "High Level Model Listing" on page 172.

2.7 High Level Plot

A plotting program was written to graphically display the system's response to inputs. The program shows the system status during a "window" of time. The position of the robot, the status of the machines and the number of pallets on the

input conveyor is all shown graphically. This allows the designer to very easily see how the system is responding without having to wade through a trace output. The plot routines are described in more detail in the section "High Level Plot" on page 41 and they are listed in the appendix "High Level Plot Listings" on page 183.

2.8 Detailed Model

The detailed model was the last part of the modeling system developed. The model developed allows the software algorithms on the system-host and mini-hosts to be verified. The utility of the communication lines in the system can be checked. The system-host and mini-hosts were modeled in detail. The axis-controller model had little detail. All of the main types of communication are modeled. Asynchronous serial communication using both ASCII and binary data formats are modeled. Interrupt and polling methods are modeled, as are software and hardware handshakes. Dedicated signal lines are also modeled.

The detailed model is described in the chapter "The Detailed Model" on page 50. The model is printed in the appendix "Detailed Model Listing" on page 226, and the supporting routines are listed in the appendix "FMS FORTRAN Listings" on page 256.

In general, the detailed models can have a varying degree of detail. The designer creates models only as detailed as needed at that time. Different models can have different parts of the system "expanded". Each detailed model should be descendent from the high level model or from one or more detailed models and should have a structure similar to the other models. This allows the high level model to aid in the verification and debugging of the detailed models. The same output routines can be used by all models, though new output that deals specifically with the detailed aspects may be developed that is not compatible with other models. For this research only one detailed model was developed. There were several intermediate models developed between the high level model and the presented detailed model. These intermediate models will not be discussed in detail in this thesis.

The system through-put, the utility of the machines, computers and communication lines can all be checked with the detailed models, though the main purpose for writing the detailed models is to develop acceptable software algorithms, which is the main motive for this research.

Chapter 3

HIGH LEVEL MODEL

This chapter describes the high level model. Included in the description of the model is how to model other FMSs with similar models. The output available, and the use of that output is given. This includes the graphical high level plot. Some sample runs are provided, indicating the kind of conclusions that can be made with the high level model.

3.1 Overview of the Model

The high level model consists of four blocks of code. The blocks represent a pallet's arrival, the logic to move a pallet to its next station, the machining stations and the exit conveyor. The flow diagram for the entire system is shown in Figure 3.1 on page 22. The blocks will be described in the next section. I will use the term "pallet", instead of "part", for the mounted work piece, because the concern here is for moving the standard size pallet through the system, not for the machining of a part.

There are two elements that are related to the model. The experiment and events. The experiment contains the definition of most of the data elements and resources used in the model. The experiment will be described in the section "The Experiment Frame" on page 32. The events are

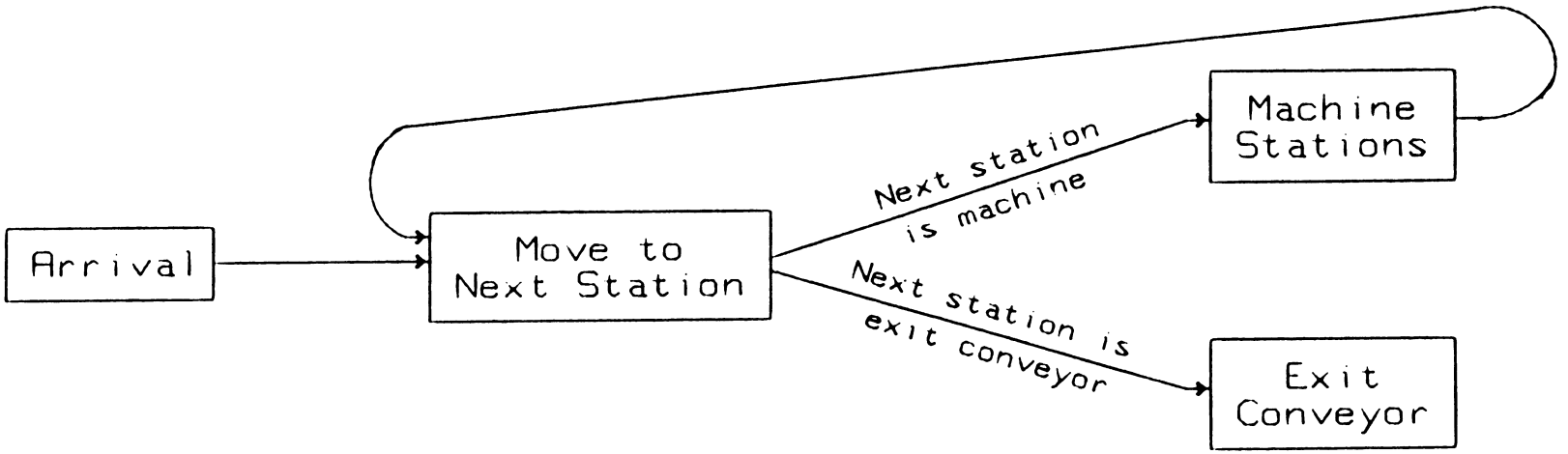

Figure 3.1 High Level Model Flow

used for a specialized trace of the system and to allow the user to change any elements in the parameter set at the beginning of each run. The trace is described in the section "The Trace" on page 38.

The time base for the model is one second.

3.2 Model Description

Before the model blocks are described, an explanation of some of the common elements of the model will be given. The stations are numbered in a clockwise order around the robot starting with the input conveyor as station one. The input and exit conveyors are considered as stations to make the movement of the pallets easier. The robot does not have a station number. Each of the stations has an indexed resource by the name MACHINE(I), where I is the station number. When a pallet needs to move to a station it must first check to see if the station has another pallet committed to it. This is indicated by the availability of the MACHINE resource. If the station is committed, a pallet is on the way to the station or a pallet is at the station.

The time the machine is busy machining a part is called the machine utility. The utility of each of the machines is stored in the global variable X(I). This value is not used to control the simulation, it is used only for data collection.

The three files that make up the model are printed in the appendix "High Level Model Listing" on page 172. The first is HL.DES which contains a description of the model and lists all of the variables used by the model. It is very important to keep a constantly updated list of the variables used. It is almost impossible to keep track of which variables are used and which variables are left for modifications and additions. Since this model will have many additions made to it when the detailed model is constructed, it is essential that the modeler knows what variables are used. The second file in the appendix is the model frame file, HL.MOD. This file contains the model statements. The last file is the experiment file, HL.EXP, which contains the experiment statements.

The trace and initialization routines used for the high level model are a subset of the routines listed in the appendix "FMS FORTRAN Listings" on page 256. The event routines are set up to be used by both the high level and detailed models. The initialization routines are listed in the file PRIME.FOR in that appendix. The trace events for the high level model are the first five events in the file TRACE.FOR.

Each of the blocks that make up the high level model will now be described.

3.2.1 Pallet Arrival Block

The pallet arrival block models the arrival of a pallet at the input conveyor. The block determines the pallet's type. The pallet waits on the conveyor until all of the pallets in front of it have been moved off the conveyor. The pallet then can attempt to move to its first station. This last step is handled by the move-to-next-station block. The flow diagram for the block is shown in Figure 3.2 on page 26.

The arrival is modeled by a CREATE block using an exponential distribution to represent the time between arrivals. An exponential distribution is used because it is assumed that each arrival is independent of every other arrival. Currently the batch size is set at one.

The pallet is next initialized. It is given a unique pallet number and its pallet type is randomly determined from a table of pallet types and the probability of a pallet being of that type.

The pallet then waits to move to the first position on the input conveyor. When the pallet moves to the first position on the input conveyor it gains control of the MACHINE(1) resource. The pallets wait to control the MACHINE(1) resource in a first-in-first-out queue that represents the conveyor.

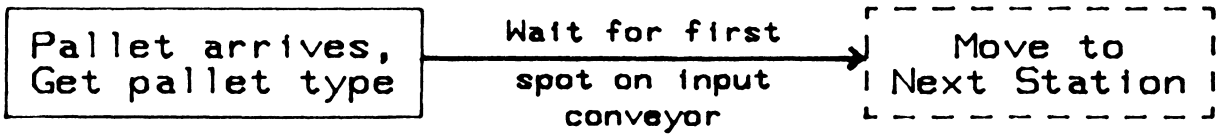

Figure 3.2 Pallet Arrival Block

Once the pallet gains control of the MACHINE(1) resource it moves to the move-to-the-next-station block.

3.2.2 Move to the Next Station Block

The move to the next station block is the most complicated block in the model. Its flow diagram is shown in Figure 3.3 on page 28. First, the pallet's next station and processing time are found, then the pallet tries to gain control of the robot, and, once it does, it moves to its next station.

How the pallet gains control of the robot is determined by the pallet movement control program in the system-host. The only time a conflict between pallets for the control of the robot can occur is after the robot has finished moving a pallet. At any other time the robot will either be busy, in which case the pallet will have to wait for the robot to finish; or the robot will be idle, which indicates that it has nothing to do and it can move the pallet right away without checking to see if another pallet wants the robot. So the only time that the pallets waiting for the robot must be ranked according to priority is when the robot finishes with a pallet.

To implement the priority scheme in the model, all of the pallets waiting for the robot are placed in a queue. The pallets are ordered in the queue by the priority they have

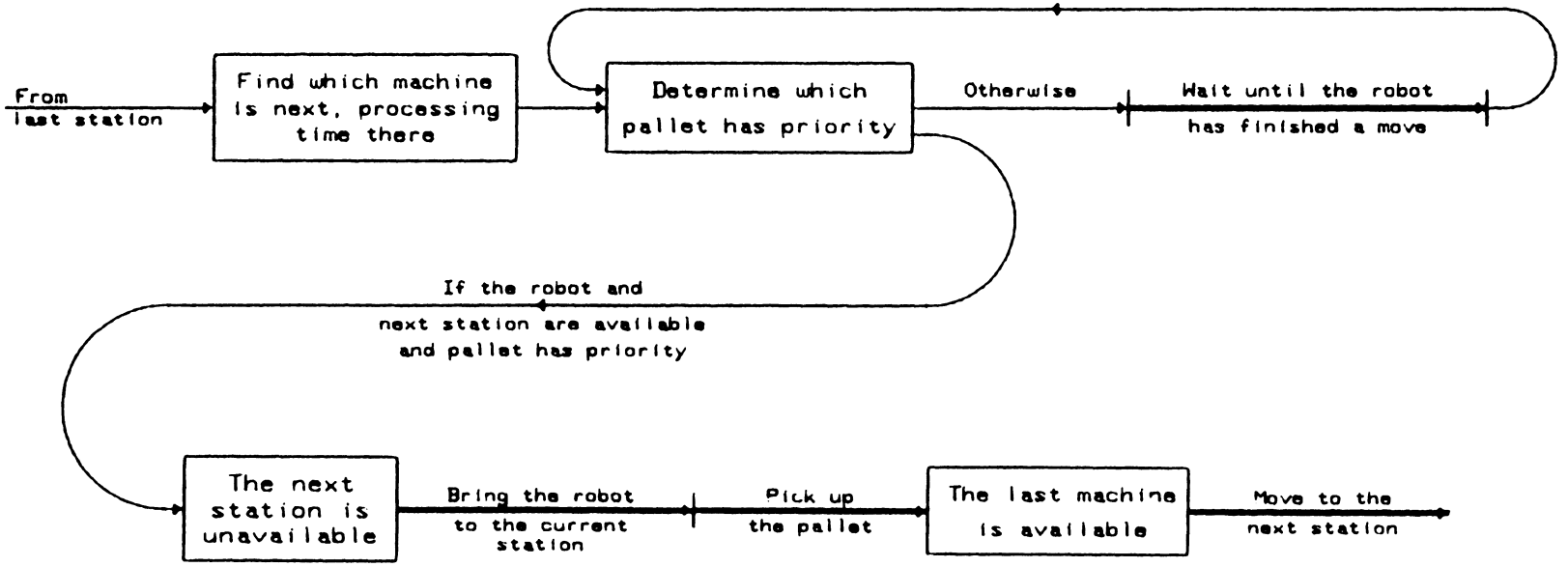

Figure 3.3 Move to the Next Station Block

in gaining control of the robot. In the system modeled, the pallet that has been in the system the longest has priority. So, the queue of waiting pallets is ordered by the time the pallet arrived in the system, which is stored in one of the pallet's attributes. For other ranking methods, a priority number would be assigned to one of the pallet's attributes and the queue would be ordered by that attribute. For a very complicated ordering method, an event may be used to directly place the pallet in the queue in the right position.

Whenever the robot is done with a pallet a signal code is sent that releases all of the waiting pallets. The pallets are checked in the order they were in the queue to see if both their next machine and the robot is available. If one or both are not, the pallet goes back to the waiting queue. Once one pallet is found that can move on, it requests the robot, setting the robot busy, so the rest of the pallets will go back to the waiting queue. If no pallet is found, the robot remains idle until a new pallet arrives or a pallet finishes processing at a machine.

Once the pallet can move on, it requests both the next station's MACHINE resource and the robot. It is guaranteed to control both resources, since they both needed to be available in order for the pallet to get to this point. The pallet will then wait for the robot to arrive at the pallet's current station. SIMAN automatically takes care of calculating the time it will take the robot to move from

where it is to the pallet's station. Once the robot is at the pallet's current station the pallet is picked up by the robot. This is modeled as a time delay. The pallet then moves to its next station. SIMAN calculates the time it will take the robot to move to the pallet's next station.

3.2.3 The Machine Station Block

The machine station block is used to model the machines. All of the machines are modeled by one SIMAN "macro". The machine the pallet is at is determined by the pallet's station number attribute, called M. All of the machine dependent functions in the block are indexed by M so that the functions of the various machines are not mixed.

The flow of the block is shown in Figure 3.4 on page 31. The pallet is placed on the machine by the robot. This is modeled as a time delay. The robot is then released and the machine is set to "busy". The processing is modeled by a time delay.

The time to bring the machine back to its "home" position is then calculated. The "home" position is the position the machine must be in when the robot places or picks up a pallet. The time to bring the machine to its home position is modeled separately from the machining time because this time is not specified by the part program. A uniform distribution is used to model this time. In

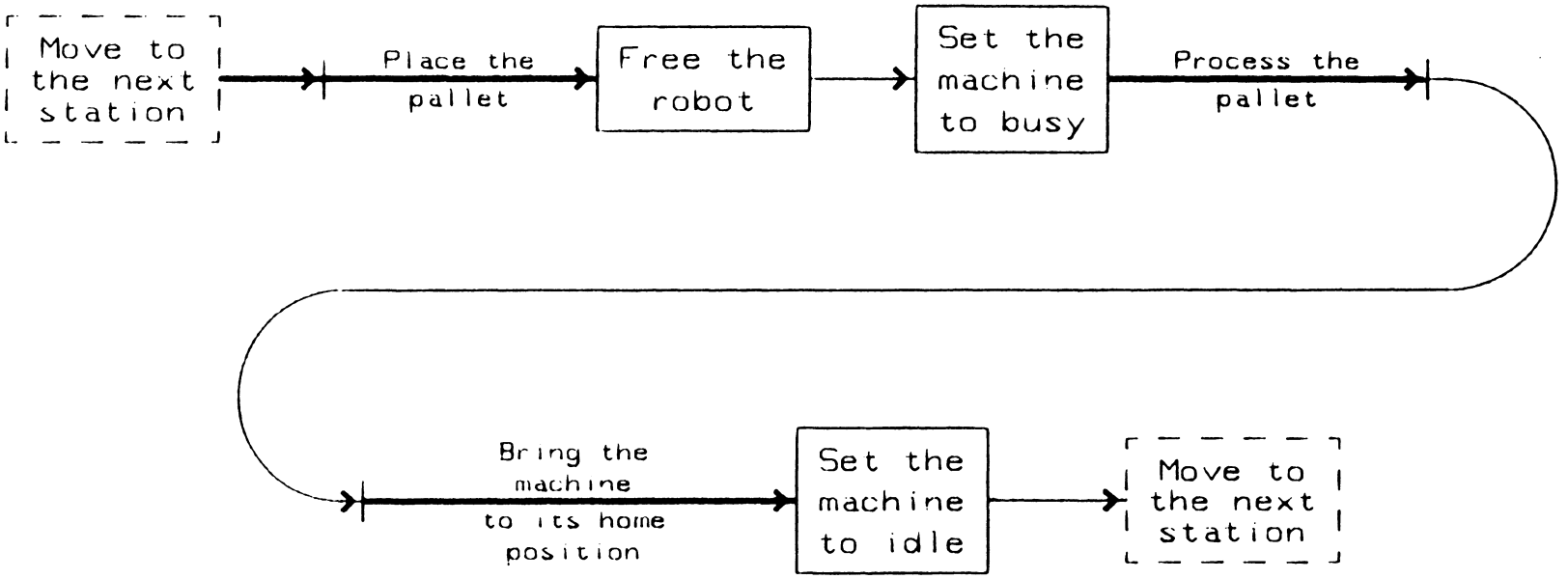

Figure 3.4 The Machine Stations

practice, this time will be almost the same for each pallet of a particular type, so the time can be included in the processing time for that pallet type on that machine. In this case the range of the delay should be set to zero or to a small value to represent minor variations between pallets.

After the machine is at its home position, the machine is set to idle and the pallet moves to the move-to-the-next-station block.

3.2.4 The Exit Conveyor Block

The exit conveyor block models the exit conveyor in the system. The flow of the block is shown in Figure 3.5 on page 33.

The pallet is placed on the exit conveyor in the same manner as it would be on a machine. After it is placed the conveyor is made available so other pallets may leave the system. Before the pallet leaves the system, data is collected about its stay in the system. Currently the only data collected is the in-system-time. The pallet then leaves the system.

3.3 The Experiment Frame

The experiment frame contains all of the values in the model that may change from run to run. This setup allows the

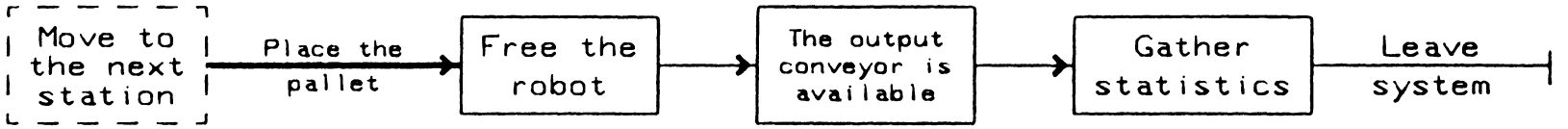

Figure 3.5 The Exit Conveyor

definition of parameters to be separate from the model, making it much easier to change the definitions.

The experiment has elements to define the maximum storage allowed for the model, and the number and length of the runs. The MACHINE resource and ROBOT are defined. The velocity of the robot is specified. The DISTANCES element defines the distance between the stations.

The PARAMETERS element is used to define the parameter sets. The use of the parameter sets is defined by the modeler. There are parameter sets that specify the time between pallet arrivals, the probability that a pallet is of a certain type, the times it takes to pick up and place a pallet and the range of times it takes each machine to move to its home position.

The machine sequence for each of the pallet types is defined in the parameter set. Each pallet type uses a parameter set. The first five parameters are reserved for the machine sequence. The end of the sequence is indicated by the pallet moving to station six, the exit conveyor. Since a pallet may not go to the same machine twice, the maximum number of machines it can move to is five, which is the amount of storage reserved for the sequence. The next five parameters are used for the fixed processing times at each machine in the sequence.

The ordering in the queues is defined in the experiment. The queue in which pallets wait for the robot is ordered

based on the lowest value first of the attribute that contains the time the pallet arrived at the system.

The random number streams can be initialized in the experiment. A different stream should be used for each random variable so different versions of the model when run with the same seeds will have the same random number sequence, allowing easy comparison between the two versions. This is especially useful for verifying the detailed model with the high level model.

The standard output, trace output and disk output are all specified in the experiment. The output available is described in the following sections.

3.4 Standard SIMAN Output

The standard output from SIMAN is in the form of a summary report. What is included on this report is specified in the experiment frame. A sample summary report is shown in Figure 3.6 on page 36. The report is broken into four parts. The first part is an identifier which identifies which run of what model the report is for. The next three parts are used to display data gathered during the run.

The Tally section is used to display data that is considered to be independent observations. The statistics are based on the number of observations, not on the time the observations were made. In the high level model this is used

SIMAN Summary Report

Run Number 1 of 1

Project: FMS HL MV2.18 EV2.20
Analyst: Tim Martin
Date : 5/ 7/1985

Run ended at time : .7200E+04

Tally Variables

Number	Identifier	Average	Standard Deviation	Minimum Value	Maximum Value	Number of Obs.
1	IN SYSTEM TIME	1542.45	900.29	392.42	3573.64	17

Discrete Change Variables

Number	Identifier	Average	Standard Deviation	Minimum Value	Maximum Value	Time Period
1	Station A Util.	.22	.41	.00	1.00	7200.00
2	Station A Commit	.48	.50	.00	1.00	7200.00
3	Station B Util.	.15	.36	.00	1.00	7200.00
4	Station B Commit	.53	.50	.00	1.00	7200.00
5	Station C Util.	.36	.48	.00	1.00	7200.00
6	Station C Commit	.59	.49	.00	1.00	7200.00
7	Station D Util.	.00	.00	.00	.00	7200.00
8	Station D Commit	.00	.00	.00	.00	7200.00
9	Robot Utilizatn	.66	.48	.00	1.00	7200.00
10	# on Conveyor	5.65	5.14	.00	17.00	7200.00
11	#Wait4 Rbt & Stn	.90	.85	.00	3.00	7200.00
12	Robot Position	3.77	1.68	1.00	6.25	7200.00

Counters

Number	Identifier	Count	Limit
1	PALLET NUMBER	35	Infinite

Figure 3.6 SIMAN Standard Summary Report

to collect the time a pallet spent in the system. Each time a pallet leaves the system its in-system-time is recorded. This value is useful for determining the system through-put. Generally, the lower the average time in the system the higher the through-put will be. The final measure of through-put is the number of pallets that were completely processed in a simulation. However, it is easier to compare the average in-system-time than the through-put because the through-put depends more on the input sequence than the average time does.

The Discrete Change section displays values that change discretely. The statistics are based on time. These values are used to determine the utility of the various elements in the system. This is helpful in finding if the machines are being used enough and to pinpoint any bottlenecks in the system.

Several of the variables listed need explanation. The "# on Conveyor" variable indicates the number of pallets on the input conveyor. The "#Wait4 Rbt & Stn" variable is the number of pallets waiting for the robot or their next station. This is the length of the waiting queue, mentioned in the section "Move to the Next Station Block" on page 27. The last variable, "Robot Position", does not represent accurately the robot's average position. The variable is used only for plotting, and to plot it, it must be written

to disk during the run. To write it to disk, it must be included on the summary report.

All of the variables in the Discrete Change section, except "#Wait4 Rbt & Stn", can be written to disk during the run so that they can be plotted after the simulation is done. The plotting is described in the section "High Level Plot" on page 41.

The last section on the summary report is the Counters section. The counter is used to number each pallet as it enters the system. This is used for identification of the pallet and is especially useful when following the trace output, which is described in the next section. The value on the summary report is the number of pallets that entered the system during the run. The difference between this and the number of observations for the in-system-time is the number of pallets left in the system at the end of the run. This tells if the system was over-loaded during the run.

3.5 The Trace

The trace output is generated by events that are invoked from the model frame. An event is a FORTRAN subroutine that is used to help model the system. The calls to the trace event are put at important transition points in the model.

The model has five trace events. Each one is called from only one point in the model. The trace events are used

when a pallet arrives at the system, when a pallet is ready to move to the next station, when a pallet is picked up by the robot, when the pallet arrives at a machine and when a pallet leaves the system. The trace prints the time and the number of the pallet causing the trace. The situation occurring is printed, including the status of the system. The part of the system status that is printed is dependent upon the trace event. The status of the system can be the number of pallets on the input conveyor, which machines are unavailable and which pallets are waiting for the robot or their next machine. A sample trace is shown in Figure 3.7 on page 40.

The trace can be turned on and off at specified times. One of the parameter sets is used to specify the initial trace condition and a list of the times the trace is to be turned on or off. The trace may be turned on or off any number of times.

The trace is very useful for debugging the system. It tells how each pallet is moving through the system, so it is easy to tell when the pallet does not move as expected. This also isolates the error to one of the blocks of code in the model, so it is fairly easy to pinpoint most problems using the trace. SIMAN provides a standard trace that can be used to check every operation that is done in the model to further pinpoint the error.

```
Time:  736.6 Pallet  1 was done at Machine 4, ready to move to Machine 6
               Unavailable machines:  4

Time:  736.6 Pallet  1 was picked up by the robot, is on route to Machine 6

Time:  815.2 Pallet  1 left the system.  It was in the system  815.2 sec.
               Pallets waiting for the robot or a machine:
                    None
               All machines are available

Time: 1457.8 Pallet  2 arrived.  Pallet type:  4, Machine Sequence:  4-6
               There are  1 pallet(s) on the input conveyor

Time: 1457.8 Pallet  2 was done at Machine 1, ready to move to Machine 4
               All machines are available

Time: 1474.2 Pallet  2 was picked up by the robot, is on route to Machine 4

Time: 1569.3 Pallet  2 arrived at Machine 4, Processing Time will be 240.0 sec
               Pallets waiting for the robot or a machine:
                    None
               Unavailable machines:  4

Time: 1737.5 Pallet  3 arrived.  Pallet type:  3, Machine Sequence:  3-6
               There are  1 pallet(s) on the input conveyor

Time: 1737.5 Pallet  3 was done at Machine 1, ready to move to Machine 3
               Unavailable machines:  4

Time: 1786.8 Pallet  3 was picked up by the robot, is on route to Machine 3

Time: 1809.3 Pallet  2 was done at Machine 4, ready to move to Machine 6
               Unavailable machines:  3  4

Time: 1865.5 Pallet  3 arrived at Machine 3, Processing Time will be 120.0 sec
               Pallets waiting for the robot or a machine:
                    Pallet  2 waiting for machine 6
               Unavailable machines:  3  4

Time: 1881.9 Pallet  2 was picked up by the robot, is on route to Machine 6

Time: 1960.6 Pallet  2 left the system.  It was in the system  502.8 sec.
               Pallets waiting for the robot or a machine:
                    None
               Unavailable machines:  3

Time: 1985.5 Pallet  3 was done at Machine 3, ready to move to Machine 6
               Unavailable machines:  3

Time: 1998.5 Pallet  4 arrived.  Pallet type:  4, Machine Sequence:  4-6
               There are  1 pallet(s) on the input conveyor

Time: 1998.5 Pallet  4 was done at Machine 1, ready to move to Machine 4
               Unavailable machines:  3  6

Time: 2000.1 Pallet  5 arrived.  Pallet type:  3, Machine Sequence:  3-6
               There are  2 pallet(s) on the input conveyor

Time: 2034.8 Pallet  3 was picked up by the robot, is on route to Machine 6
```

Figure 3.7 Sample Trace Output

The trace is also very useful for verifying that the model is correct, because it outputs the operation of the model in terms of the real system, not the SIMAN system. The trace's detail is such that the simulation flow can be followed without being overloaded with details, allowing someone that is not familiar with the model to be able to evaluate the output to see if it correctly represents the way the real system should work.

The trace can also be used to evaluate the response of the system. Where the system is performing less than optimally can be found and diagnosed with the trace. In many cases the trace will be too detailed to help with this. In this case, the plot program presented in the next section should be used. The trace can be used to provide the details at critical points found by looking at the plot output.

3.6 High Level Plot

The plot routine is run after the SIMAN simulation has finished. The data saved on disk during the run must be converted to ASCII format by the SIMAN output processor before it can be used by the plot program.

The plot program allows the status of the system to be displayed versus time. The time window may be changed interactively to any set of times. A plot may be presented in either color or monochrome on a CRT and may be printed with

a graphics printer or pen plotter. A sample plot is shown in Figure 3.8 on page 43. The lines on the plot at the level of the input conveyor represent the number of pallets waiting on the input conveyor. For each pallet waiting on the conveyor, there is one line. The lines at the level of the machine represent the machine utility. The solid lines represent when the machine is busy with a part and the dotted lines represent when the machine has a pallet committed to it. When there is no line, the machine is available. The line that moves from machine to machine represents the robot's position. When the line is solid the robot is moving with a pallet. When the line is dashed the robot is moving to pick up a pallet, and when it is dots the robot is idle.

The plot is used for the same reasons the trace is. It can help debug the model, help verify the model and help indicate possible improvements to the system. The trace and plot should be used together. The plot provides a quickly comprehended representation of the system from which general trends can be found. The trace provides the details about what and why the system is doing what it is doing. Together, they provide an easy way to comprehend how the system is performing and how it can be improved.

The plot routines are listed in the appendix "High Level Plot Listings" on page 183.

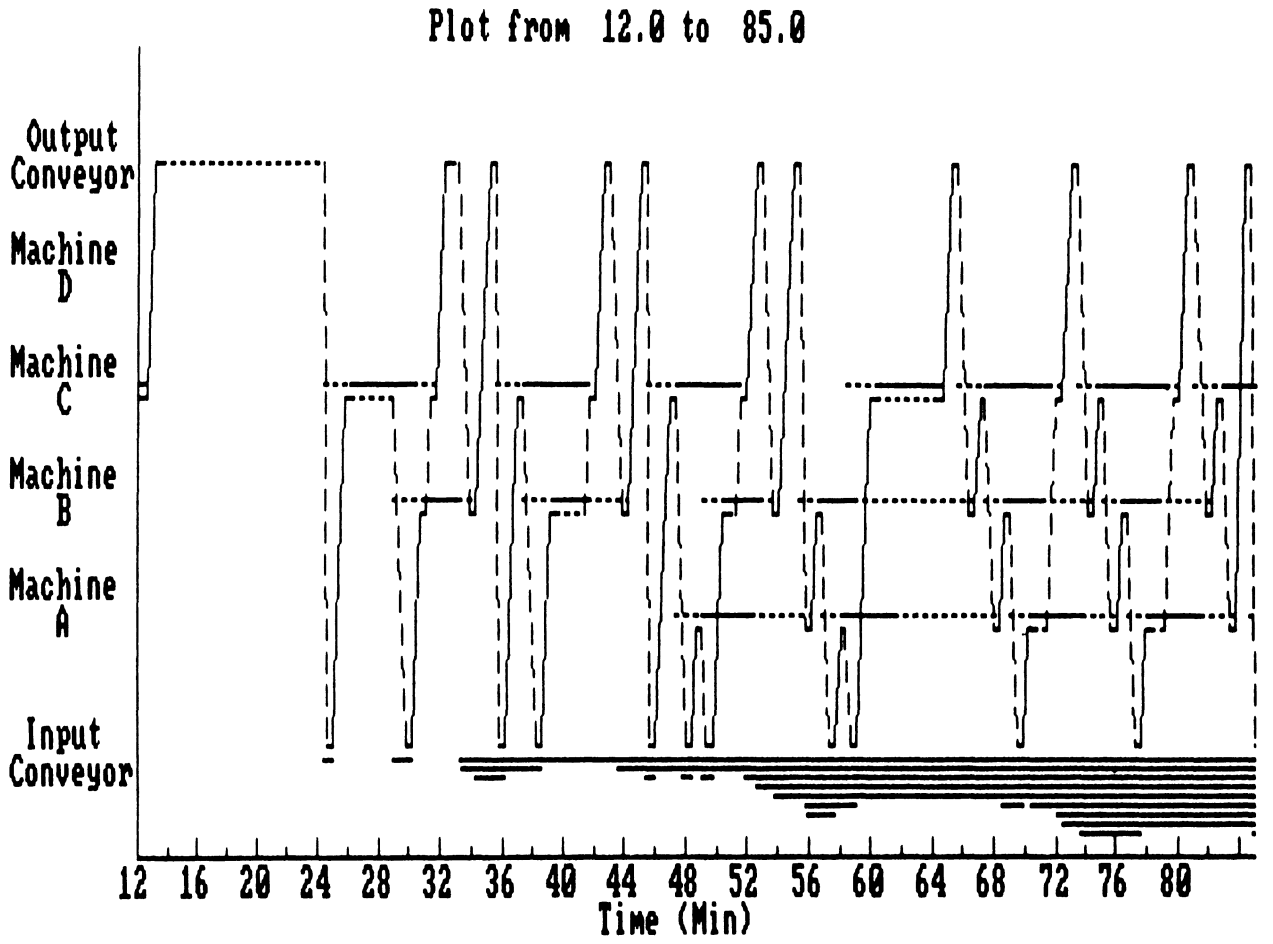

Figure 3.8 Sample Plot Output

3.7 Verification of the High Level Model

The model was verified in two ways. The first way was to examine the trace and plot to determine if the model was performing as expected. This is basically part of the debug step. The second verification is much harder to perform normally. This is to check to see if the correctly working model accurately represents the real system. At this point in the design there usually is not a real system to compare the model to, so experts that were not involved in the model development are usually called in to evaluate the output of the model to see if it represents the way the real system will work. For the model developed for this research, the real system already existed, so the model output was checked against the performance of the real system. The two agreed to the accuracy of the high level model. The model performed slightly faster because the disk access and software times are not modeled in the high level model.

3.8 Modifying the Model

It is quite easy to modify the model to try out new configurations. Almost all of the modifications will only have to alter one of the blocks of code in the model. An example is that if a pallet had a choice of machines to move to next, only the move-to-next-station block would need to

be modified. Another example is that if room for the storage of one pallet is added at each machine station, only the machine station block would have to be modified. This change is shown in Figure 3.9 on page 46. Each added line is indicated by an asterisk at the beginning. This would be a very major hardware change to the real system, but it is a minor change to the model.

3.9 Sample Runs

A sample run has been shown in the examples of the standard summary report, trace output and plot output. (See Figure 3.6 on page 36, Figure 3.7 on page 40 and Figure 3.8 on page 43). This output shows that the machines are under utilized, with more time spent waiting to have a pallet removed from a machine than is spent in processing. This indicates that some storage should be included at the machines so the pallet can wait at the machine, without stopping the machine from processing another part. This is the modification mentioned in "Modifying the Model" on page 44. The summary report from the modified model run with the same input as the non-modified model run is shown in Figure 3.10 on page 47. This shows that the storage improves the performance of the system considerably, with about 20% improvement in the though-put and machine utilization.

```
 ;
 ;
 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
 ;                                                                            ;
 ;                                                                            ;
 ;          The machine stations                                              ;
 ;                                                                            ;
 ;                                                                            ;
 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
 ;
        STATION, 2-5;
        ASSIGN: X(7) = M;        The robot is at the current station
        ASSIGN: A(3) = M;        Set A(3) to the current station
 ;
        DELAY: P(4,2);           Preform a place
        ASSIGN: X(7) = M+.25;    The robot is done at station M
 ;
        FREE: ROBOT;             Done with the robot for now
        EVENT: 4;                Trace event 4 - arrival at a machine station
        SIGNAL: 1;               Tell the world that the robot is done
 ;
*       BRANCH, 1:-              Check to see if the machine is available
*            IF, ( X(M) .LE. 0 ), GETMACH:-Yes, get the machine
*            ELSE, WT4MACH;                No, wait for it
*;
*WT4MACH  QUEUE, 9;
*       WAIT: M:-                Wait until the machine is done
*       NEXT( STARTMAC );        Start the machine
*;
*GETMACH  DELAY: P(9,1);         Mount the pallet on the machine
 STARTMAC ASSIGN: X(M) = 1;      Set the machine to busy.  Used to find
 ;                                    the machine utilization.
        DELAY: A(4);             Processing time
 ;
        ASSIGN: A(6) = M + 3;    Set A(6) to the parameter set that has
 ;                                    the data on how long it takes to
 ;                                    bring the machine to home
        DELAY: UN( A(6), 2);     Delay by the time it takes to bring the
 ;                                    machine to its home position
*       DELAY: P(9,1);           Unmount the pallet
*       SIGNAL: M;               Tell any waiting pallet that it was just
*;                                    mounted
        ASSIGN: X(M) = 0:-       Done with the machine
  -
        NEXT( MOVEON );          Move on to the next station
```

Figure 3.9 Modified Machine Station Block to Include Storage for one Pallet

SIMAN Summary Report

Run Number 1 of 1

Project: FMS HL MV2.19 EV2.20
Analyst: Tim Martin
Date : 5/ 7/1985

Run ended at time : .7200E+04

Tally Variables

Number	Identifier	Average	Standard Deviation	Minimum Value	Maximum Value	Number of Obs.
1	IN SYSTEM TIME	1297.12	666.37	484.11	2802.70	21

Discrete Change Variables

Number	Identifier	Average	Standard Deviation	Minimum Value	Maximum Value	Time Period
1	Station A Util.	.32	.47	.00	1.00	7200.00
2	Station A Commit	.70	.79	.00	2.00	7200.00
3	Station B Util.	.25	.43	.00	1.00	7200.00
4	Station B Commit	.65	.83	.00	2.00	7200.00
5	Station C Util.	.51	.50	.00	1.00	7200.00
6	Station C Commit	.91	.70	.00	2.00	7200.00
7	Station D Util.	.00	.00	.00	.00	7200.00
8	Station D Commit	.00	.00	.00	.00	7200.00
9	Robot Utilizatn	.80	.40	.00	1.00	7200.00
10	# on Conveyor	3.69	3.41	.00	11.00	7200.00
11	#Wait4 Rbt & Stn	.85	.84	.00	3.00	7200.00
12	Robot Position	3.61	1.71	1.00	6.25	7200.00

Counters

Number	Identifier	Count	Limit
1	PALLET NUMBER	35	Infinite

Figure 3.10 Modified Model Output

Other changes that might improve the system performance can be deduced from the output. From the plot, it is obvious that the robot should not sit at the output conveyor after dropping off the last pallet in the system. It should move to the input conveyor to be ready to pick up the next arriving pallet. This can be generalized to anytime the robot is idle, it may want to move to a position that is more likely to be closer to the next station it will have to move to. Further analysis is needed to reach any conclusions about this.

Also from the plot, it can be seen that sometimes the system has an all machines busy, no machines busy cycle. This may be due to the ordering of which pallet is to move first. The pallets that have been in the system longer and are moving to the exit conveyor have priority over pallets that are moving to an idle machine. By giving higher priority to a pallet moving to an idle machine the system performance may improve. Again, a series of simulations is necessary to determine if an alternate ordering scheme would improve system performance.

Many more similar types of conclusions can be drawn from the high level model.

Even though a modified system would perform much better than the original system, the detailed model will be of the original system, so that it will model the physical system. Normally the best configuration would be found for the system

and then only the best configuration would be modified into the detailed model. For this research, the "best" configuration is considered to be the configuration that exists.

Chapter 4

THE DETAILED MODEL

This chapter describes the detailed model. The relationship between the detailed model and the high level model is explained. Special attention is given to the modeling of polling and interrupt driven processes.

4.1 Overview of the Model

The model is contained in six files. The FMS.DES file contains a description of the model and lists all of the variables used and how they are used. The FMS.MOD file contains the SIMAN model frame representing most of the model. The FMS.EXP file contains the SIMAN experiment frame, which defines most of the variables used in the model. The last three files were written in FORTRAN. After they are compiled, they are linked to the SIMAN run processor routines to form a new run processor. This new run processor can be used by both the high level and detailed models. The EVENT.FOR file contains the non-trace events used by the model. The PRIME.FOR file contains the initialization routines and wrap-up routine called before and after each run. The last file, the TRACE.FOR file contains the trace events. The FMS.DES, FMS.MOD and FMS.EXP files are printed in the appendix "Detailed Model Listing" on page 226, and the

EVENT.FOR, PRIME.FOR, and TRACE.FOR files are listed in the appendix "FMS FORTRAN Listings" on page 256.

The model has nine blocks, which are shown in Figure 4.1 on page 52. The connections between the blocks indicate how the blocks can interact with each other and do not represent program flow. This figure shows how complex the interactions are between the elements in the system. The simulation is necessary to check that all of the parts of the system communicate correctly with each other.

All of the main routines in each of the processors is modeled as a separate block. The system-host polling routine block represents the system-host polling routine. The system-host interrupt routine block represents the interrupt routines used in communication with the mini-hosts. The mini-host polling routine block and the axis-controller block represent the software on those processors.

In addition to the blocks that totally represent software, some software is represented by the blocks that control the physical functions of the system. The software represented by these blocks is run on the system-host. The blocks in this category are the pallet arrival, pallet processing, machine processing and exit conveyor blocks.

The last block, the task list block, is used to represent the task list on the system-host. All of the blocks will be described in the section "Description of the Model Blocks" on page 58.

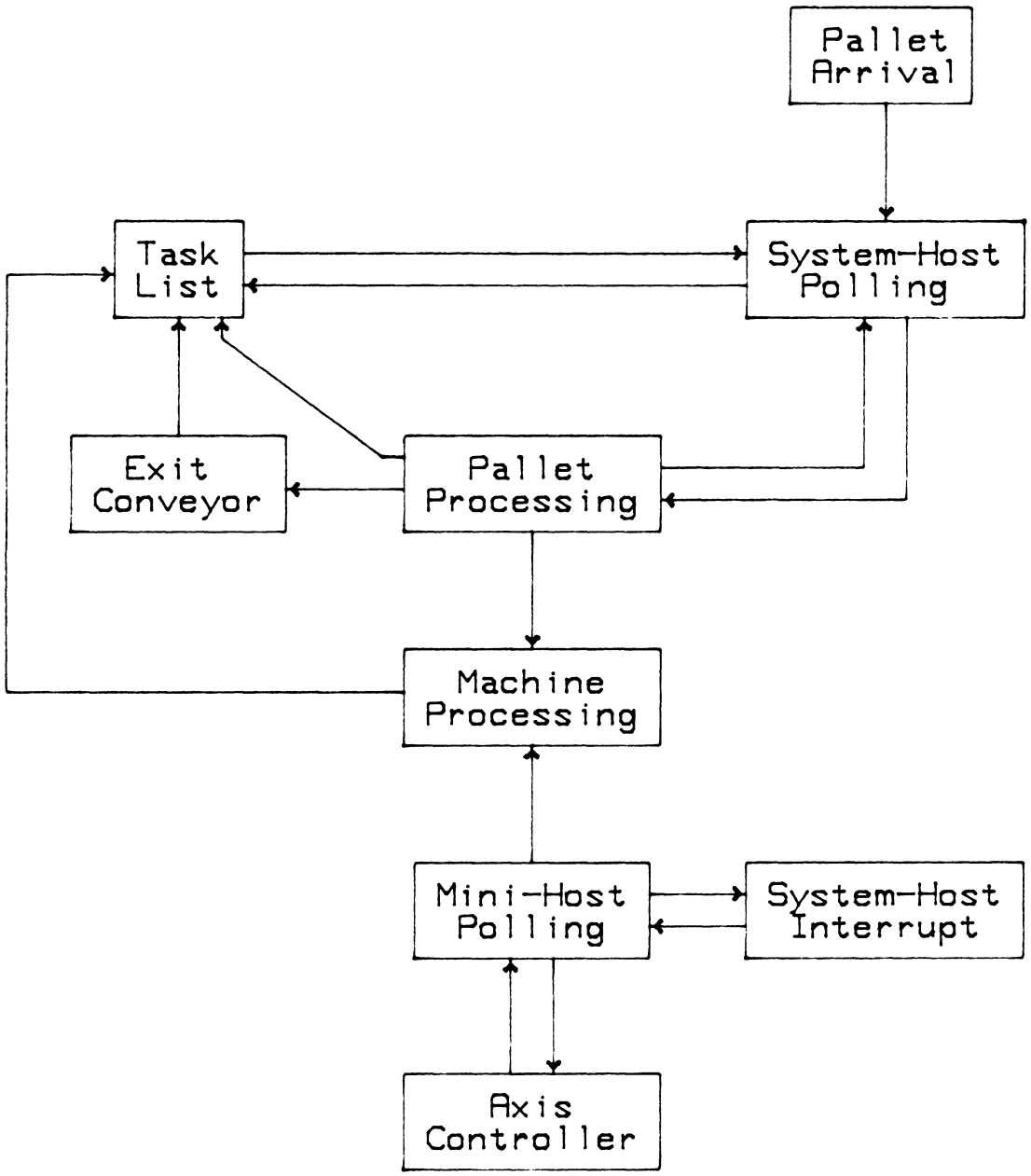

Figure 4.1 Detailed Model Blocks and Their Connections

Unlike the high level model, which is always "controlled" by the pallet entity, the detailed model has different types of entities "controlling" different parts of the model. The entity that moves through the model during the simulation "controls" the simulation. Each of the software blocks is controlled by a separate entity that represents the program flow in that routine. All of the rest of the blocks are controlled by pallet entities.

Each computer has a resource that represents the use of the Central Processing Unit (CPU) of the computer. Any time a routine needs to use the CPU it must first gain control of the CPU resource. This guarantees that the CPU will not be used by more than one routine at a time.

The basic numbering scheme for the computers is different than the station numbering used in the high level model. For purposes of movement, the numbering of the stations is the same, with the machine stations numbered two through five and the input and exit conveyors numbered one and six, respectively. Since the conveyors do not have computers and the robot does, number one is used to represent the robot computers and number six is not used. The numbering of the machine station computers is the same as the machine station numbering for movement. There are many cases where groups of indexed variables are used for the computers. For these the numbering consists of the base numbering plus

some multiple of five. This makes it easier to decipher which variable goes with which computer.

4.2 Representing Computer Storage

I have not modeled the overall storage on any computer. I assume that the overall storage will be laid out when the program is coded and that there will be no problems fitting the programs and data in memory. A scheme that keeps track of how the total memory space is used could probably be developed using the techniques used here to keep track of specific parts of memory. However, for a control computer, most of the memory will remain static and only specific parts of the memory need to be checked for overflow. That is how the memory structure of the computers has been modeled.

Standard files (queues) in SIMAN are used to model the memory. The amount of memory taken by the data will be proportional to the length of the file. Each time something is stored in the file the file length can be checked to see if there is room for the data in memory, and if not, the processing can be altered to take care of the situation.

4.3 The Modeling of Polling and Interrupt Routines

Both polling and interrupt routines were modeled. The actual implementation of the two methods was very similar.

To model an interrupt routine, the interrupt routine entity waits in a queue until an interrupt occurs. The interrupt is modeled using a signal code, which releases the waiting entity from the queue. The entity then requests the CPU at the interrupt's priority. This may take the CPU away from another entity. Once the CPU is obtained, the interrupt routine is executed. At the end of the interrupt, there must be a check to see if another interrupt occurred, and if one did, it must be processed. If not, the entity is placed back in its queue waiting for another interrupt.

To enable and disable interrupts, another queue is used to hold the entity when the interrupts are disabled. When they become enabled a signal code is sent that releases the entity to check to see if an interrupt occurred. If one hasn't it moves to the wait for interrupt queue. To disable interrupts, the interrupt routine entity is removed from the wait for interrupt queue and is sent to the disabled queue.

Modeling polling uses the same structure as modeling interrupts. The reason polling is not modeled as a constant checking of the status of the polled variable is that it would take way too long to simulate this. There are still two ways left to model polling. The first is to use an activity scanning approach and check the status of the polled variable anytime any variable in the system changes. Since the status is checked very often with this method, it leads to long run times. However, this is a very good way to model

polling, so it should be considered when developing a polling model.

The last way to model polling is the way that was used in this research. The polling routine is modeled like an interrupt routine, except that the CPU is requested at the lowest priority when the polled status changes. The condition that changes the polled status must send a signal code to tell the polling entity that the status has changed. A variable delay time may be added after the signal code is sent to represent the time it will take the polling routine to recognize the change in status. Most polling routines are so short that this will not be necessary.

There are some cases where a polling routine is used to delay for an almost fixed time, such as the transmission of a character. In these cases a simple time delay is used to model the polling routine.

The main problem with the interrupt and polling modeling methods described is that the condition must be checked before the entity is placed in the queue to wait for the signal code. If it is not checked, and the condition is true, the condition will not be processed until the next time the conditions is set true. The additional check does not correspond to anything that is in the software. A method to eliminate the need for or to hide this extra check is needed.

4.4 Description of the Events Used

Several events were written to aid in the design of the model. Events were used either to simplify the model or to perform a task that is impossible to do from the model frame.

Three of the events are closely related. The send-message and message- arrive events are used to model a Universal Asynchronous Receiver/Transmitter (UART), which is used to send messages over an asynchronous serial line. The receive-message event is used to read the message sent. The send-message event is passed the transmission time for the message, where the message is to be sent, over what line the message is to be sent and the message to be sent. It schedules the message- arrive event to occur after the transmission time. The message-arrive event stores the message and sends the requested signal code. The receive-message event is used by the receiving processor to read the message (bring it out of storage). This set of three events makes it easy to model the communication lines.

The update-the-task-list event is used to do just that. The event is entered by a pallet that needs its entry on the task list updated so that it agrees with the pallet's status. If the pallet is not on the task list, it is added to it. If the pallet is on the task list, the entry is updated to agree with the pallet's attributes. This event is very

difficult to model in the model frame, but straight forward in the event frame.

The last event that is used to help model is the communication error event. It is used to print an error message if a "bad" character is received by the system-host. This event is impossible to model in the model frame because messages cannot be printed from the model frame.

The rest of the events are used for the trace or for initialization. These events are described elsewhere.

4.5 Description of the Model Blocks

In this section each of the model blocks will be described. How the blocks relate to the high level model and how they can be generalized will be explained.

4.5.1 The Pallet Arrival Block

The pallet arrival block is the same as in the high level model, except that the pallet does not move to the move-to-next-section block. Instead, a message is sent to the system-host telling it that a pallet arrived and telling it what type of pallet it is. The flow diagram for the block is shown in Figure 4.2 on page 59.

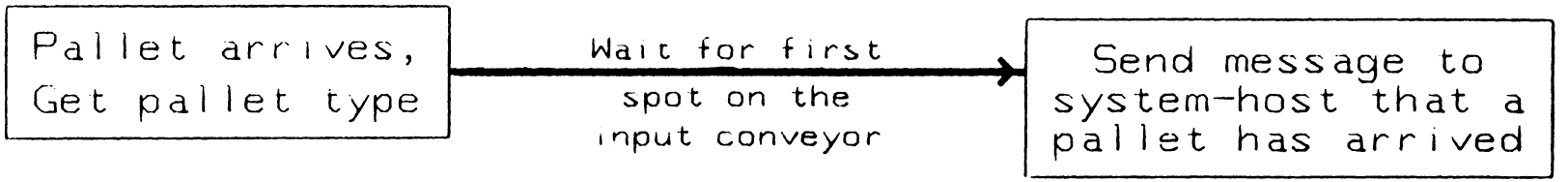

Figure 4.2 The Pallet Arrival Block Flow

4.5.2 The Task List Block

The task list block is simply a queue, there is no flow. The task list is an array that has one entry for each pallet in the system. For this purpose, the input conveyor is not considered part of the system. The first pallet on the input conveyor can be on the task list if at least one machine in the system is uncommitted. The task list is used to keep track of the status of the pallets in the system. The priority the pallet has is determined by where it is on the task list. The task list is always ordered by when the pallet arrived to the system. The ordering is taken care of automatically by SIMAN. The wait for the robot queue of the high level model roughly represents the task list. The main difference is that the task list contains all of the pallets in the system and the wait for the robot queue only has the pallets that are waiting for the robot or a machine.

4.5.3 The System-Host Polling Routine Block

The system-host polling routine is responsible for controlling what pallet goes to what machine. This represents part of the move-to-next-station block in the high level model. The polling routine is broken into two main parts, a part controlled by the system-host polling entity, and a part controlled by the pallets. The part controlled

by the polling entity is described in this section. The part controlled by the pallets is described in the next section, "The Pallet Processing Block" on page 64. The flow for the part of the polling routine controlled by the polling entity is shown in Figure 4.3 on page 62.

The first thing the routine polls for is to see if a pallet has arrived to the system. If one has, it is placed in the input queue. Placing the pallet in the input queue is taken care of in the pallet arrival block. If there are any pallets in the input queue, the polling routine checks to see if there is room on the task list, and if so, it moves the pallet from the input queue to the task list. The REMOVE command and a software delay are used to model this.

When no more pallets can be added to the task list, the task list is checked to see if there are any idle pallets that can move to their next station. A SEARCH command is used to check for an idle pallet which is indicated by one of the pallet's attributes. If an idle pallet is found, it is sent to the pallet processing block for processing. The polling entity waits in a queue until the processing is done.

The pallet processing block is responsible for removing the polling entity and sending it to the right place in the model when the processing block is done. If the pallet could not move to its next station, the polling entity is sent to check the task list for another idle pallet. If none is

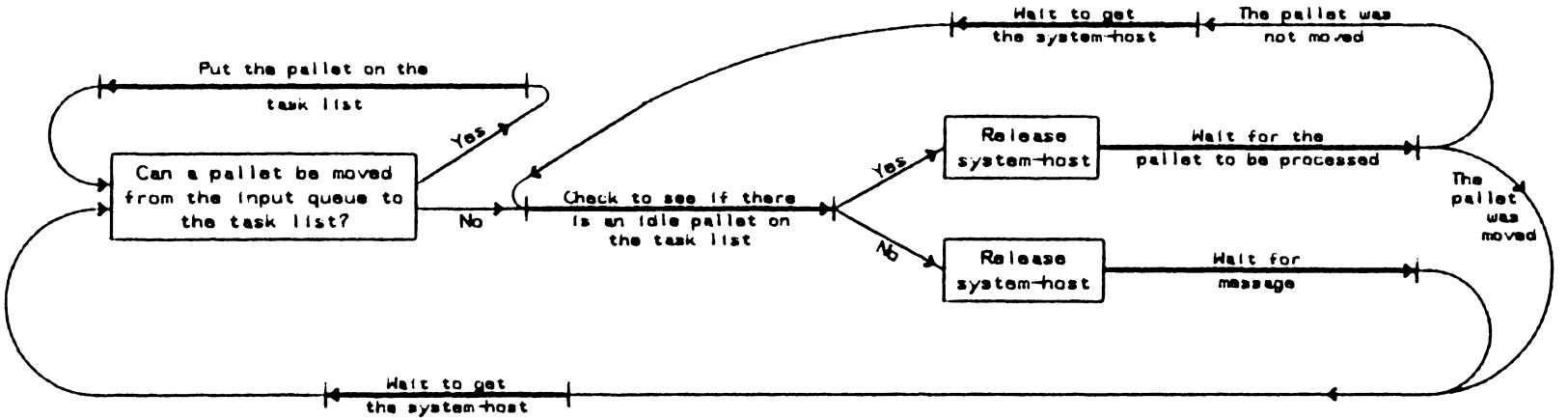

Figure 4.3 System Host Polling Routine Flow

found, the entity waits for another pallet to arrive or for a process to finish to start the polling again.

If the idle pallet could move on, then the polling entity waits for the pallet processing block until the pallet arrives at its next machine. The polling entity is then sent to the top of the polling routine.

The system host resource is needed whenever any of the polled conditions is checked. The resource is acquired at lowest priority at the beginning of the polling routine. When the control passes to the pallet processing block, the resource is released so the pallet processing block can use it. The resource is also released before the polling entity goes into the queue to wait for a message to be sent it. As a result, most of the polling time is not counted as part of the computer utility.

The technique of placing an entity in a queue to wait for a part of a routine that is controlled by another entity is very useful for coordinating a complicated routine that is easier to model using several blocks controlled by different entities. When the control passes from one block to the next, the entity for the next block is removed from its queue and sent to the appropriate place in its block. The entity from the last block is then put in a queue until its block is needed again.

4.5.4 The Pallet Processing Block

The pallet processing block represents part of the system-host polling routine. It is used to move a pallet to its next station, if possible. The flow of the block is shown in Figure 4.4 on page 65.

The first step in the pallet processing is updating the task list, if necessary, so the pallet's next station and processing time will be correct. To indicate that a pallet has already been updated, it's idle attribute is set to 0.5 from 0. When the pallet is busy on a machine, the idle attribute is set to 1. This update may be better done in the machine station block after the machine processing is done. It is in the pallet processing block because that is where it was in the high level model. In a further version of the detailed model, it may be moved.

The next step is to see if the pallet's next station is available. If it is not, the pallet is placed back in the task list and control returns to the system-host polling routine block. If the pallet can move on, then the robot and machine are requested and the pallet is moved to its next station. This is modeled almost identically to the last part of the move-to-next-station block in the high level model. The only difference is that the time it takes to calculate a robot move has been added.

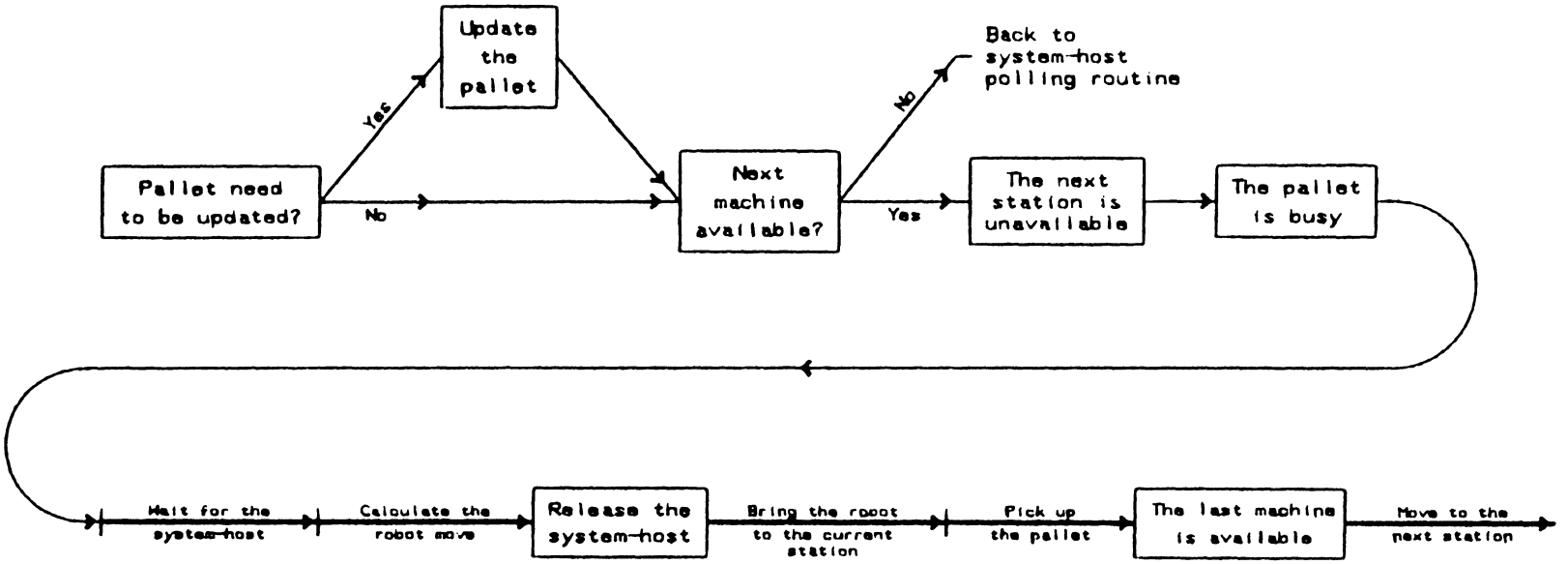

Figure 4.4 The Pallet Processing Block Flow

4.5.5 The Machine Processing Block

The machine processing block is used to model the pallet's and system-host's responsibility for the machine processing. This block resembles the machine station block of the high level model. The flow for the block is shown in Figure 4.5 on page 67.

The pallet is placed at the machine in the same manner as the high level model. The task list is updated to indicate that the pallet is at its new station.

The pallet is delayed by the time it takes to read the part program from the system-host's disk to the system-host's memory. This is another part of the system-host polling routine, so the system host resource is requested before the disk read is done.

The serial receive interrupt from the appropriate mini-host is enabled to allow the transmission of the part program to the mini-host.

The pallet then waits until the process is done. When the process is done the pallet is set to idle and the system-host polling entity is "woken up" so it will poll the system status again.

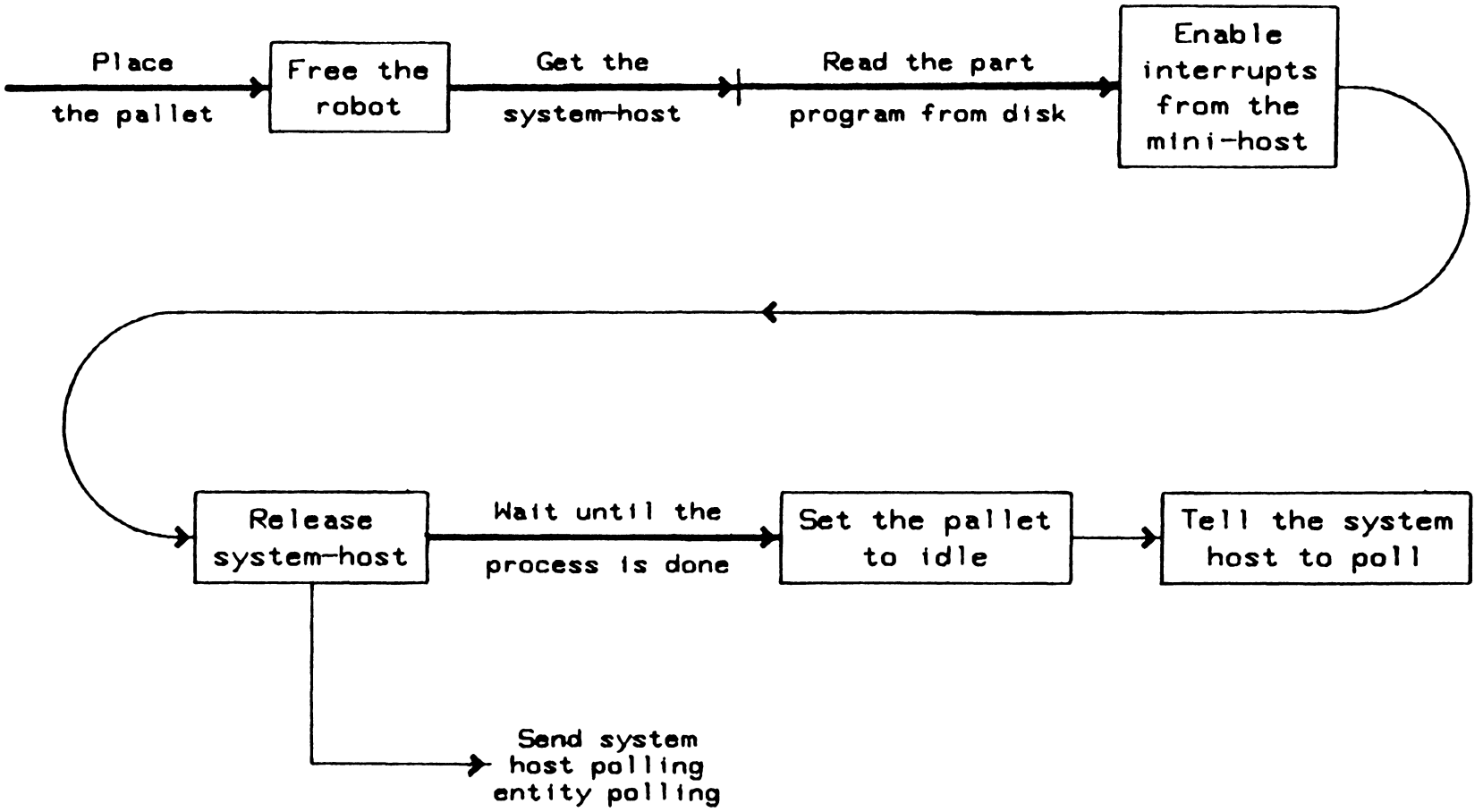

Figure 4.5 The Machine Processing Block Flow

4.5.6 The Exit Conveyor Block

The exit conveyor block is essentially the same as for the high level model. The only difference is that the pallet must be removed from the task list before it leaves the system. The flow is shown in Figure 4.6 on page 69.

4.5.7 The System-Host Interrupt Routine

The system-host interrupt routines are used to communicate with the mini-hosts. There are as many interrupt routines as there are mini-hosts. Each mini-host has a separate serial line to the system-host. Whenever a mini-host sends data to the system-host, the interrupt routine is entered. The system-host uses polling while in the interrupt routine to send data to the mini-host. The flow of the routine is shown in Figure 4.7 on page 70.

This block uses the interrupt structure, and the read and transmit events that were discussed previously. This block shows the ease of complex branching in SIMAN.

The flow consists of waiting for an interrupt, reading the character that caused the interrupt and processing the character. An XON indicates that the mini-host is ready-to-receive. The mini-host has enough storage for six commands, so a maximum of six commands can be sent after an XON. An XOFF indicates that the mini-host buffer is full;

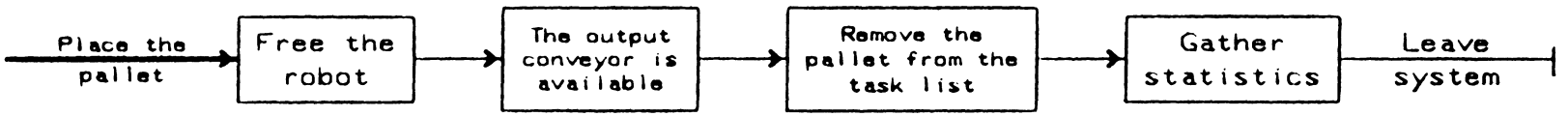

Figure 4.6 The Exit Conveyor Block Flow

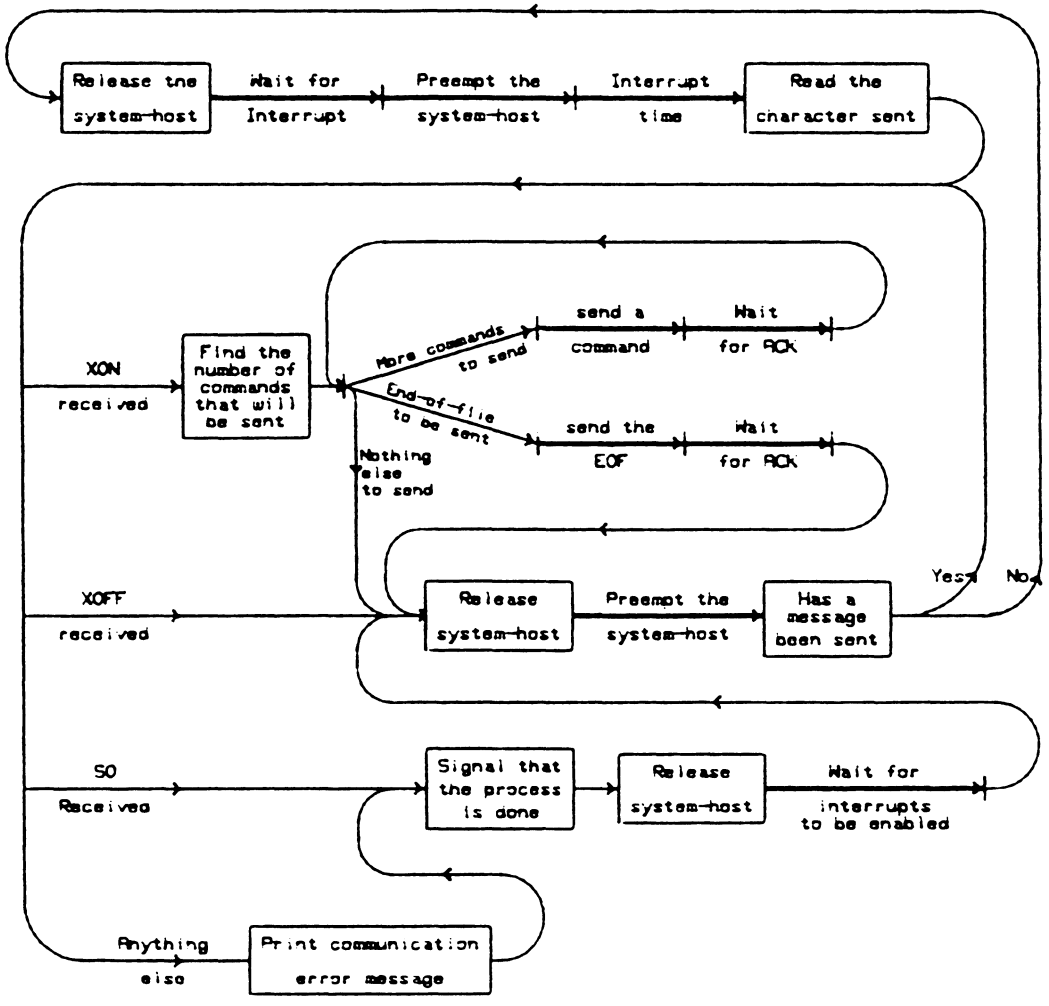

Figure 4.7 The System-Host Interrupt Routine Flow

stop transmitting. An SO indicates that all of the axis controllers have finished processing the entire part program. The serial interrupts are disabled after an SO is input. Any other character received is an error.

The commands are modeled as being sent as one unit, including a checksum. The mini-host sends an ACK or NAK to acknowledge or not the checksum received. In the present model, it is assumed that an acknowledgement will be sent, but it would be easy to add a check for a NAK, and if received, to model a resend of the command.

4.5.8 The Mini-Host Model

The mini-host has two significant routines. The main polling routine and the timer interrupt routine. The polling routine performs almost all of the processing on the mini-host. The timer routine is used to generate a clock to use for coordinating the step motors. Because of the high frequency of the clock (40kHz), the interrupt routine ends up taking a large percent (approximately 40%) of the processor time. This is why it is significant.

The interrupt routine is not modeled because it would make the simulation runs unacceptably long. Any processing delay in the model should be multiplied by the percent of the CPU time taken up by the interrupt routine. Delays for serial I/O should not be affected since the serial operation

runs in parallel with the timer and interrupts. The mini-host utility presented on the summary report will not give an accurate indication of the true CPU usage. Any unused portion of the CPU time needs to be multiplied by the percent of the CPU time taken by the interrupt routine and the product should be added to the summary report utility to obtain a better estimate of the true CPU utility.

The polling routine is responsible for receiving a part program from the system-host and sending it to the axis-controllers. The flow for the routine is shown in Figure 4.8 on page 73. The mini-host sends an XON to the system-host to indicate that it is ready to receive and then it waits to receive a command. After a command is received, the mini-host sends back an ACK. Commands are received until the mini-host buffer is full or the end of the part program is reached. A flag is set to indicate when the end of the part program is reached. The flag is modeled by using one of the attributes of the mini-host polling entity, which controls the mini-host model. Any time local storage is needed in a routine, attributes of the controlling entity can be used for the storage.

After the buffer is full or the end of the part program has been received, the mini-host sends an XOFF to the system-host to tell it to stop sending data.

The mini-host now waits for all of the axis-controllers to become ready-to-receive. The axis-controller will only

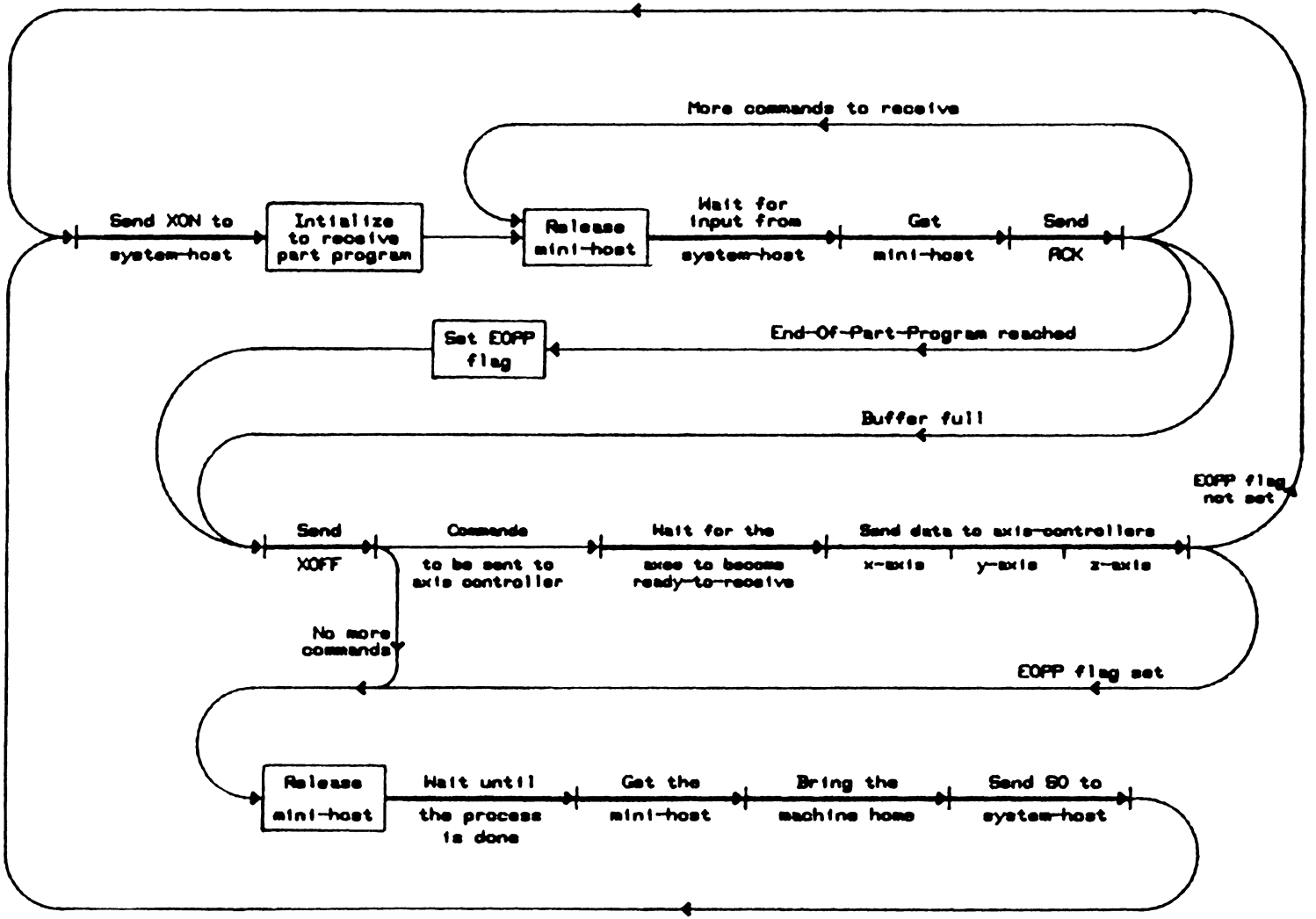

Figure 4.8 The Mini-Host Polling Routines Flow

be ready-to-receive between the processing of commands. This is modeled by a uniform delay between zero and an approximate maximum processing time for a command. In a more detailed model the mini-host would wait until a signal is sent from the axis controller. This is how a discrete signal line would be modeled. Once all of the axis-controllers are ready-to-receive, the mini-host sends the appropriate commands to each axis-controller in sequence. This is modeled all as one communication event.

If the end of the part program has not been reached, the mini-host sends another XON to the system-host and the process is repeated. If the end of the part program has been received, the mini-host waits until the axis-controllers indicate that they are done with all of the commands they have received. When the axis-controllers are done, the mini-host brings the machine home in the same manner as done in the high level model. The mini-host then sends a SO to the system-host to indicate that the process is done. An XON is then sent to indicate that the mini-host is ready to receive once the next pallet arrives.

4.5.9 The Axis-Controller Model

The part of the machine station block in the high level model where the part is actually machined is modeled by the axis-controllers in the detailed model.

The axis-controller is not modeled to the detail of the rest of the system. This shows that by using the hierarchical modeling approach, only the detail that is needed needs to be modeled. The axis-controller as it is now modeled represents all of the axis-controllers for a machine. The axis-controller could be modeled in more detail at a later date. The flow of the current axis-controller model is shown in Figure 4.9 on page 76.

The axis-controller waits for data to be sent it. Once data has been sent, the axis-controller starts processing the data. The reading of the data by the axis-controller is currently not modeled. There is no limit on the buffer size of the axis-controller in the model. The actual buffer is eighteen commands long. A more detailed model could easily model the reading of data and include the buffer length restriction.

When the axis-controller has no more commands in its buffer, it sends a signal to the mini-host to tell it it is done.

4.6 The Experimental Frame

The experiment frame is set up the same way as the experiment frame for the high level model. It is mostly just an expanded version of the high level experiment.

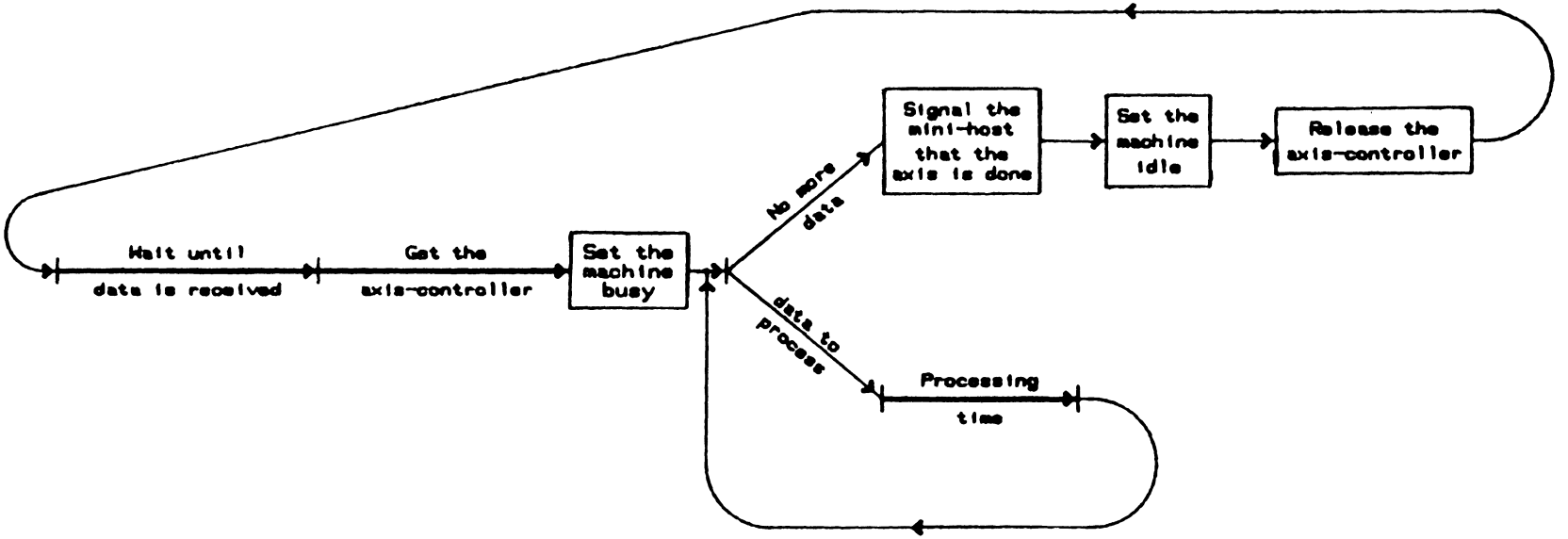

Figure 4.9 The Axis-Controller Flow

The detailed model experiment adds the definition for the CPU resources. The times for the software and hardware delays have been added to the parameter sets. A parameter set is used for the character transmission times between all of the elements in the system. Additional output commands have been added to print the utility of the additional resources modeled.

4.7 Output Available

The output available from the high level model is still available from the detailed model. The form of the trace has changed some since the modeling of the task list has made a complete list of all of the pallets in the system available.

Additional output has been provided on the standard SIMAN summary report. The utility of each of the CPU's and communication lines is now printed. It must be remembered that the mini-host utility is actually about 40% higher than the utility printed on the summary report.

Since the communication lines are full duplex, the utility in each direction is printed. The only communication lines in the system are effectively the lines into and out of each mini-host. All serial communication in the system is either to or from the mini-host. A mini-host can only talk to one element in the system at a time since its one serial

port is multiplexed. So, the limit on the communication is the mini-host serial port.

A specialized trace and plot for the detailed model would be a useful addition to the model.

4.8 Initialization of the Simulation

There are several routines that are used to initialize the system. They are listed in the file PRIME.FOR in the appendix "FMS FORTRAN Listings" on page 256. The event SETENT is also used to initialize the system.

The PRIME subroutine is called at the beginning of each run. It initializes the high level trace and allows the user to change any parameter in the parameter sets. Only values in the parameter set may be changed. To change the parameter the user types the parameter set number, the parameter number and the new value. Any number of parameters may be changed. Any change made is in effect during the entire set of runs, unless it is changed back. When the user is done making changes to the parameters, a zero is entered to start the run.

The SETENT event is used to initialize all of the CPU entities in the model.

4.9 Verification of the Model

The detailed model was verified in much the same way as the high level model. By comparing the results from the two models an additional verification is provided for both models. There have been several problems in the high level model that were noticed because of the results from the detailed models.

4.10 Modifying the Model

The detailed model can be easily modified to model different aspects of the system and to try out different software algorithms. The modifications to the detailed model are harder to do than for the high level model. In many cases more than one block will have to be changed to make the modification. An example would be adding more detail to the axis-controller model. The mini-host model would also have to be modified to model the communication between the two more accurately.

The trace events could be added easily once the critical points in the system to trace have been determined. The detailed trace should be controlled by a different set of trace on-off parameters than the high level trace.

One of the major hurdles to creating a plot routine for the detailed model is determining what values should be

plotted and how. Most of the same routines used by the high level plot could still be used for the detailed plot.

4.11 Sample Run

This section provides a sample run from the detailed model. The random number streams are the same used for the sample run for the high level model. The summary report is shown in Figure 4.10 on page 81and the revised high level trace is shown in Figure 4.11 on page 82. The high level plot looks essentially the same as the one generated with the high level model, so it is not shown.

The sample output shows that the detailed model gives results that are almost the same as the high level model, indicating that the model is functioning correctly and that software algorithms can be successfully simulated.

SIMAN Summary Report

Run Number 1 of 1

Project: FMS MV1.23 EV1.13
Analyst: Tim Martin
Date : 5/ 7/1985

Run ended at time : .7200E+04

Tally Variables

Number	Identifier	Average	Standard Deviation	Minimum Value	Maximum Value	Number of Obs.
1	IN SYSTEM TIME	1587.69	929.33	397.76	3670.73	17

Discrete Change Variables

Number	Identifier	Average	Standard Deviation	Minimum Value	Maximum Value	Time Period
1	Station A Util.	.21	.41	.00	1.00	7200.00
2	Station A Commit	.48	.50	.00	1.00	7200.00
3	Station B Util.	.15	.36	.00	1.00	7200.00
4	Station B Commit	.54	.50	.00	1.00	7200.00
5	Station C Util.	.34	.47	.00	1.00	7200.00
6	Station C Commit	.59	.49	.00	1.00	7200.00
7	Station D Util.	.00	.00	.00	.00	7200.00
8	Station D Commit	.00	.00	.00	.00	7200.00
9	Robot Utilizatn	.65	.48	.00	1.00	7200.00
10	# on Conveyor	5.76	5.23	.00	18.00	7200.00
11	Task List Length	2.14	1.19	.00	4.00	7200.00
12	Robot Position	3.79	1.66	1.00	6.25	7200.00
13	System Host Util	.02	.15	.00	1.00	7200.00
14	Mini Host 1 Util	.00	.06	.00	1.00	7200.00
15	Mini Host 2 Util	.01	.11	.00	1.00	7200.00
16	Mini Host 3 Util	.01	.09	.00	1.00	7200.00
17	Mini Host 4 Util	.03	.17	.00	1.00	7200.00
18	Mini Host 5 Util	.00	.00	.00	1.00	7200.00
19	Line to MiniHo 1	.00	.00	.00	.00	7200.00
20	Line to MiniHo 2	.00	.03	.00	1.00	7200.00
21	Line to MiniHo 3	.00	.03	.00	1.00	7200.00
22	Line to MiniHo 4	.00	.05	.00	1.00	7200.00
23	Line to MiniHo 5	.00	.00	.00	.00	7200.00
24	Line fr MiniHo 1	.00	.00	.00	1.00	7200.00
25	Line fr MiniHo 2	.00	.02	.00	1.00	7200.00
26	Line fr MiniHo 3	.00	.02	.00	1.00	7200.00
27	Line fr MiniHo 4	.00	.04	.00	1.00	7200.00
28	Line fr MiniHo 5	.00	.00	.00	1.00	7200.00

Counters

Number	Identifier	Count	Limit
1	PALLET NUMBER	35	Infinite

Figure 4.10 Sample Detailed Model Summary Report

```
Time:  750.1 Pallet  1 was picked up by the robot, is on route to Machine 6

Time:  828.8 Pallet  1 left the system.  It was in the system  828.8 sec.
               The Task List:
                    Empty
               Pallets in the input queue: None
               All machines are available

Time: 1457.8 Pallet  2 arrived.  Pallet type:  4, Machine Sequence:  4-6
               There are  1 pallet(s) on the input conveyor

Time: 1457.8 Pallet  2 was done at Machine 1, ready to move to Machine 4
               All machines are available

Time: 1474.2 Pallet  2 was picked up by the robot, is on route to Machine 4

Time: 1569.3 Pallet  2 arrived at Machine 4, Processing Time will be 240.0 sec
               The Task List:
                    Pallet  2 is busy on machine 4
               Pallets in the input queue: None
               Unavailable machines:  4

Time: 1737.5 Pallet  3 arrived.  Pallet type:  3, Machine Sequence:  3-6
               There are  1 pallet(s) on the input conveyor

Time: 1737.5 Pallet  3 was done at Machine 1, ready to move to Machine 3
               Unavailable machines:  4

Time: 1786.8 Pallet  3 was picked up by the robot, is on route to Machine 3

Time: 1819.2 Pallet  2 was done at Machine 4, ready to move to Machine 6
               Unavailable machines:  3  4

Time: 1863.5 Pallet  3 arrived at Machine 3, Processing Time will be 120.0 sec
               The Task List:
                    Pallet  2 waiting for machine 6
                    Pallet  3 is busy on machine 3
               Pallets in the input queue: None
               Unavailable machines:  3  4

Time: 1884.6 Pallet  2 was picked up by the robot, is on route to Machine 6

Time: 1963.2 Pallet  2 left the system.  It was in the system  505.5 sec.
               The Task List:
                    Pallet  3 is busy on machine 3
               Pallets in the input queue: None
               Unavailable machines:  3

Time: 1990.8 Pallet  3 was done at Machine 3, ready to move to Machine 6
               Unavailable machines:  3

Time: 1998.5 Pallet  4 arrived.  Pallet type:  4, Machine Sequence:  4-6
               There are  1 pallet(s) on the input conveyor

Time: 1998.5 Pallet  4 was done at Machine 1, ready to move to Machine 4
               Unavailable machines:  3  6

Time: 2000.1 Pallet  5 arrived.  Pallet type:  3, Machine Sequence:  3-6
               There are  2 pallet(s) on the input conveyor

Time: 2040.1 Pallet  3 was picked up by the robot, is on route to Machine 6

Time: 2059.7 Pallet  6 arrived.  Pallet type:  4, Machine Sequence:  4-6
               There are  3 pallet(s) on the input conveyor
```

Figure 4.11 Sample Detailed Model High Level Trace

Chapter 5

CONCLUSIONS AND RECOMMENDATIONS FOR FURTHER RESEARCH

5.1 Conclusions

This thesis presents a method to simulate a Flexible Manufacturing System (FMS). By using the building blocks presented, an FMS may be easily simulated both on a macro and micro scale. The macro model is used for the configuration design of the FMS. The micro model is used to help design and debug the computer hierarchy and its software.

The use of simulation to aid in the design of the FMS configuration is common. Using simulation to help design the software is rare. With the hierarchical modeling method presented, the model that would normally be developed when designing the FMS configuration needs only to be enhanced to allow the software to be modeled. This saves much time over developing a separate model for software development.

There are other advantages to the hierarchical approach. The models at different levels in the hierarchy help verify each other. The models only need to be as detailed as needed at any point in the FMS design, allowing more general problems to be solved first, without needing to worry the more detailed problems until later in the design.

The SIMAN simulation language is a good base for the modeling. Its structure and special functions make it easy to develop the hierarchical model. The built in material handling functions provide one of the building blocks needed for the model.

By using the structure of the high level model presented, a high level model of a new system can be quickly and easily developed. The modular structure of the model makes it easy to make changes, so the best configuration can be found.

The various forms of output combine to allow easy understanding of how the system is performing and how the performance can be improved.

Many aids are provided by this research in developing the detailed models. The basic communication techniques are provided, as well as examples of program flow. These aids, along with the hierarchical approach, make it much simpler to model the software design of a hierarchy of computers.

The detailed model allows the software to be checked for correct operation. It allows different algorithms to be tried. The complexity of the software can be estimated from the model. Any problems with an overloaded CPU or communication line can be found before the computers are bought.

Once the simulation has shown that the software works, the system may be built and coded with the confidence that

no major changes will have to be made to the system to make it perform well. The start-up time will be much less since the software has already been tested.

The model can still be used after the system is operational to check any enhancements before they are made to the system.

5.2 Suggestions for Further Research

Even though the modeling method presented is very useful for the design of an FMS and its software, there are still enhancements that would make future modeling easier and more effective. Some of these enhancements are described below.

The presented system should be modeled in complete detail so that others that wish to design to a detail greater than that presented will not be on their own. This includes modeling the axis controller's software and modeling all of the possible error conditions. A continuous model should be added to show that the machine control can be simulated with the same framework as the rest of the FMS.

There are several elements that are awkward to model. When modeling a polling routine a better way to guarantee that an event that is polled for is not missed needs to be developed. A straight forward way to model a latching interrupt is also needed.

A trace and plot routine designed specifically for the detailed model would aid in the verification and modification of the software algorithms.

A set of plot routines that would allow specialized plots to be very easily generated would save the designer much work in creating a plot. An interactive routine to help design the plot would be ideal.

A method to generalize this work to allow any system of hierarchical computers to be simulated easily would be the ultimate goal of continuing this research.

REFERENCES

[1] Harrington, Joseph. Computer Integrated Manufacturing, Industrial Press, New York, 1973, pp. 73-75.

[2] Vasilash, Gary S. "The Advent of Numerical Control, 1951-1959", Manufacturing Engineer, Vol. 88, No. 1, January 1982, pp. 143-172.

[3] Harrington, p. 62.

[4] Hartwig, Glenn C. "Electronic Control Moves into the Ascendancy, 1960-1969", Manufacturing Engineer, Vol. 88, No. 1, January 1982, pp. 181-202.

[5] Pressman, Roger S. and Williams, John E. Numerical Control and Computer-Aided Manufacturing, John Wiley and Sons, New York, pp. 289-291.

[6] Ibid., pp. 292-297.

[7] Harrington, p. 75.

[8] Vasilash, Gary S. "The Road to the Automatic Factory, 1970-1981", Manufacturing Engineer, Vol. 88, No. 1, January 1982, pp. 209-252.

[9] Cook, N. H. "Computer Managed Parts Manufacture", Scientific American, 22-29, February 1975.

[10] Hutchinson, G. K. and Wynne, B. E. "A FMS", IE, Vol. 5, No. 12, December 1973, pp. 10-17.

[11] Talvage, Joseph. "Simulating Manufacturing Systems", IEEE Spectrum, Vol. 20, No. 5, May 1983, pp. 53-55.

[12] Olig, Gene A. "An Overview of an FMS System", IEEE Transactions on Industry Applications, Vol. IA-21, No. 2, March/April 1985, pp. 318-323.

[13] Iwata, K. "FMS Software - The Present and Future", Technocrat, Vol. 15, No. 9, September 1982, pp. 32-44.

[14] Hart, James Allen. "Miniature Automated Flexible Manufacturing System", Master's Thesis, VPI&SU, Blacksburg, Virginia, October 1983.

[15] Draper, Charles Stark Laboratory. Flexible Manufacturing System Handbook, Noyles Publications, Park Ridge, New Jersey, 1984, p. 152.

[16] HEI Corporation. "The HEI Real Time Computer Simulator", Product literature, HEI Corporation, Carol Stream, Illinois, 1985.

[17] Draper, Charles Stark Laboratory. Flexible Manufacturing Systems Handbook, Noyles Publications, Park Ridge, New Jersey, 1984, pp. 69-122. This gives the entire procedure for buying an FMS.

[18] Rose, Lawrence L. "Hierarchical Modeling in GASP", 14th Annual Simulation Symposium, IEEE Computer Society and ACM, 1981, pp. 199-213.

[19] Pegden, C. Dennis. Introduction to SIMAN, Systems Modeling Corporation, State College, Pennsylvania, 1982.

[20] Wysk, R. A., et al. "Control System Design and Analysis for Flexible Manufacturing Systems", VPI&SU, Blacksburg, Virginia, October, 1980.

[21] Nunnally, Charles E. and Ellis, George H. "Smart Stepper Motor Control", 12th Annual Southeastern Symposium on System Theory, IEEE Computer Society, 1980, pp. 194-196.

[22] Chang, Tien-Chien, et al. "Simulation of a Microprocessor - Based CNC Machining System", VPI&SU, Blacksburg, Virginia, June, 1980.

[23] Nunnally, C. E. and Wysk, Richard A. "The use of Iconic Models for Automated Manufacturing Instruction", 89th ASEE Annual Conference, Los Angeles, June 1981, Vol. 3, pp. 825-830.

[24] Wysk, R. A., et al. "Physical Machining Systems at Virginia Polytechnic Institute and State University", Annual Industrial Engineering Conference Proceedings, Institute of Industrial Engineers, 1982, pp. 398-404.

[25] Hart, James Allen. "Miniature Automated Flexible Manufacturing System", Master's Thesis, VPI&SU, Blacksburg, Virginia, October, 1983.

[26] Nunnally, C. E. and Hart, J. A. "Miniature - Automated Flexible Manufacturing System", 2nd Annual Workshop on Interactive Computing: CAD/CAM: Electrical Engineering Education, George Washington University, Washington, D. C., November 30 - December 2, 1983.

[27] Nunnally, C. E. and Hart, J. A. "Multiaxis Hierarchical Control", Southeastcon, IEEE, 1984.

[28] Hart, p. 3.

[29] Pritsker, A. Alan B. and Claude Dennis Pegden. Introduction to Simulation and SLAM, Halsted Press, New York, 1979, p. 2.

[30] Ibid., p. 6.

[31] Ibid., p. 68.

[32] Most of the information in this and the following sections comes from Pegden, C. Dennis, Introduction to SIMAN, 1982, and "SIMAN Microcomputer Installation Manual". These are both published by the distributor of SIMAN, Systems Modeling Corporation, Calder Square, P.O. Box 10074, State College, Pennsylvania 16805.

[33] Pegden, C. Dennis. Introduction to SIMAN, Systems Modeling Corporation, State College, Pennsylvania, 1982, p. 20.

[34] Ibid., pp. 20, 40.

[35] Ibid., pp. 8, 9, 65.

[36] Ibid., pp. 4, 5, 251-256.

[37] Ibid., pp. 6, 251-256.

[38] Ibid., pp. 7, 251-256.

[39] Ibid., pp. 64, 68.

[40] Ibid., p. 257.

[41] Ibid., pp. 226, 258.

[42] Ibid., p. 237.

[43] Ibid., p. 197. Pegden references Felberg, E., "Low-Order Classical Runge-Kutta Formulas with Step-Size Control and Their Application to some Heat Transfer Problems",

NASA Report TR R-315, Huntsville, Alabama, April 5, 1969

BIBLIOGRAPHY

American Society of Tool and Manufacturing Engineers. Introduction to Numerical Control in Manufacturing. Edited by Raymond E. Howe, American Society of Tool and Manufacturing Engineers, Dearborn, Michigan, 1969.

Barnes, Michael F. Measurement and Modeling Methods for Computer Systems Performance Studies, Langton Information Systems, England, 1979.

Bell, Thomas E. "Objectives and Problems in Simulating Computers", AFIPS Conference Proceedings, Vol. 41, Part 1, 1972, pp. 287-297.

Chang, Tien-Chien, et al. "Simulation of a Microprocessor - Based CNC Machining System", VPI&SU, Blacksburg, Virginia, June, 1980.

Cook, N. H. "Computer Managed Parts Manufacture", Scientific American, 22-29, February 1975.

Draper, Charles Stark Laboratory. Flexible Manufacturing System Handbook, Noyles Publications, Park Ridge, New Jersey, 1984.

Fleming, Robert H. and Berry, Margaret E. "Hierarchical Multi-Granular Modeling to Improve Software Performance", 15th Annual Simulation Symposium, IEEE Computer Society and ACM, 1982, pp. 223-238.

Haigh, Peter L. "A Cost Effective Modeling Technique for polling Systems", 14th Annual Simulation Symposium, IEEE Computer Society and ACM, 1981, pp. 227-267.

Haigh, Peter L. "A Methodology for Simulating Computer Systems", 15th Annual Simulation Symposium, IEEE Computer Society and ACM, 1982, pp. 277-313.

Harrington, Joseph. Computer Integrated Manufacturing, Industrial Press, New York, 1973.

Hart, James Allen. "Miniature Automated Flexible Manufacturing System", Master's Thesis, VPI&SU, Blacksburg, Virginia, October 1983.

Hartwig, Glenn C. "Electronic Control Moves into the Ascendancy, 1960-1969", Manufacturing Engineer, Vol. 88, No. 1, January 1982, pp. 181-202.

Heck, Robert. "Simulation: A Tool for Evaluating Computer Systems Architecture", 15th Annual Simulation Symposium, IEEE Computer Society and ACM, 1982, pp. 91-98.

HEI Corporation. "The HEI Real Time Computer Simulator", Product literature, HEI Corporation, Carol Stream, Illinois, 1985.

Hutchinson, G. K. and Wynne, B. E. "A FMS", IE, Vol. 5, No. 12, December 1973, pp. 10-17.

Iwata, K. "FMS Software - The Present and Future", Technocrat, Vol. 15, No. 9, September 1982, pp. 32-44.

Machinability Data Center. Machining Data Handbook, Second Edition, Machinability Data Center, Metcut Research Associates, Cincinnati, 1972.

Microcompatibles. "Grafmatic, FORTRAN Graphics for the IBM Personal Computer", Microcompatibles, 1983.

Microcompatibles. "Plotmatic Documentation", Microcompatibles, 1984.

Nunnally, Charles E. and Ellis, George H. "Smart Stepper Motor Control", 12th Annual Southeastern Symposium on System Theory, IEEE Computer Society, 1980, pp. 194-196.

Nunnally, C. E. and Hart, J. A. "Miniature - Automated Flexible Manufacturing System", 2nd Annual Workshop on Interactive Computing: CAD/CAM: Electrical Engineering Education, George Washington University, Washington, D. C., November 30 - December 2, 1983.

Nunnally, C. E. and Hart, J. A. "Multiaxis Hierarchical Control", Southeastcon, IEEE, 1984.

Nunnally, C. E. and Wysk, Richard A. "The use of Iconic Models for Automated Manufacturing Instruction", 89th ASEE Annual Conference, Los Angeles, June 1981, Vol. 3, pp. 825-830.

Nutt, Gary. "A Case Study of Simulation as a Computer System Design Tool", Computer, Vol. 11, No. 5, pp. 31-36, October 1978.

Olig, Gene A. "An Overview of an FMS System", IEEE Transactions on Industry Applications, Vol. IA-21, No. 2, March/April 1985, pp. 318-323.

Pegden, C. Dennis. Introduction to SIMAN, Systems Modeling Corporation, State College, Pennsylvania, 1982.

Pressman, Roger S. and Williams, John E. Numerical Control and Computer-Aided Manufacturing, John Wiley and Sons, New York.

Pritsker, A. Alan B. and Claude Dennis Pegden. Introduction to Simulation and SLAM, Halsted Press, New York, 1979.

Rose, Lawrence L. "Hierarchical Modeling in GASP", 14th Annual Simulation Symposium, IEEE Computer Society and ACM, 1981, pp. 199-213.

Salomon, Fred A. and Tafuri, Debra A. "Emulation - A Useful Tool in the Development of Computer Systems", 15th Annual Simulation Symposium, IEEE Computer Society and ACM, 1982, pp. 55-71.

Scott, Harold R. "Design and Evaluation Methodology for Computer-Controlled Manufacturing Systems", PhD. Dissertation, VPI&SU, Blacksburg, Virginia, December 1982.

Systems Modeling Corporation. "SIMAN Microcomputer Installation Manual", Systems Modeling Corporation, State College, Pennsylvania.

Talvage, Joseph. "Simulating Manufacturing Systems", IEEE Spectrum, Vol. 20, No. 5, May 1983, pp. 53-55.

Texas Instruments Incorporated. MS-FORTRAN User's Guide and Reference Manual (MS-FORTRAN 3.0), Texas Instruments Incorporated, Houston, Texas, 1982.

Vasilash, Gary S. "The Advent of Numerical Control, 1951-1959", Manufacturing Engineer, Vol. 88, No. 1, January 1982, pp. 143-172.

Vasilash, Gary S. "The Road to the Automatic Factory, 1970-1981", Manufacturing Engineer, Vol. 88, No. 1, January 1982, pp. 209-252.

Williams, Theodore J. "The Development of Reliability in Industrial Control Systems", IEEE Micro, Vol. 4, No. 6, December 1984, pp. 66-80.

Wysk, R. A., et al. "Control System Design and Analysis for Flexible Manufacturing Systems", VPI&SU, Blacksburg, Virginia, October, 1980.

Wysk, R. A., et al. "Physical Machining Systems at Virginia Polytechnic Institute and State University", Annual Industrial Engineering Conference Proceedings, Institute of Industrial Engineers, 1982, pp. 398-404.

Appendix A

INTRODUCTION TO SIMAN

This appendix gives an overview of the SIMAN language. Background information on computer simulation modeling is provided to improve the reader's understanding of SIMAN and how SIMAN fits into the computer modeling world. The appendix describes the SIMAN language and how to run SIMAN simulations. I evaluate the features of SIMAN as they are described. The appendix ends with an overall evaluation of SIMAN.

A.1 Introduction to Computer Simulation Modeling

This section gives an overview of computer simulation modeling. The difference between modeling and simulation is discussed. The types of models that can be simulated on a digital computer are described. The choice of the type of language to use for the model is discussed.

A.1.1 Modeling and Simulation

Computer simulation modeling is the process of developing a model that is to be simulated on a digital computer. A **model** is a description of a real, or hoped to be real, system. A model can be scaled physical objects,

mathematical relations or graphical representations [29]. **Simulation** is the process of representing the dynamic behavior of a system by moving the model of the system from state to state [30].

In order for a digital computer to simulate the model, the model must be in a form that can be understood by a simulation program. There are many computer languages available that a computer simulation model can be written in. These vary from general purpose languages to general purpose simulation languages to special purpose simulation languages. In all of these, to allow the states of the system to be changed by a computer, the system states must be represented by a set of variables.

The output of the simulation does not give an analytical relationship between the inputs and outputs. Any analytical relationship must be derived from analysis of the output. This is like a laboratory experiment. Therefore, simulation runs are often called **experiments**.

A.1.2 Modeling Orientations

A computer simulation model can be continuous, discrete or both. In continuous models the states of the system can change continuously with time. In discrete models the states of the system only change at discrete instants in time. There are three methods that are commonly used to implement

discrete models. The methods are called event, activity and process modeling. These three methods, combined with continuous modeling, form four modeling methods, called modeling **orientations**. I will describe each of the orientations in more detail below.

A.1.2.1 Continuous Modeling

Continuous models are usually a set of equations of some sort. The next state of the system is represented by mathematical relationships that involve the present and past states of the system. The relationships often only specify the derivative of the next state, in which case the simulator must integrate the derivatives to find the next state.

The usual case is that the modeler specifies the next state as first order derivatives of a combination of the current states of the system. Some simulation languages allow more complex relationships to be modeled directly. Higher order differential equations can be used or frequency domain equations may be used.

There are some discrete models that are modeled much as the continuous systems above. These models can be thought of more as having a continuous orientation than a discrete orientation. It is sometimes easier to simulate models such as these using a continuous orientation simulator rather than

a discrete orientation simulator. Not all continuous simulators can handle discrete changes, though.

A.1.2.2 Discrete Modeling

The manner in which the different discrete orientations are modeled varies quite a bit. In event modeling any event that causes a change in the system must be modeled. The events that make up the model are independent from each other, except that one event may schedule another event to occur. In activity modeling, each activity is modeled separate from every other activity. An activity involves a start and stop event. The conditions that cause an activity to start or stop are specified in the activity. Each activity is independent from every other activity. In process modeling the entire process or system is modeled as one unit, and each of the parts of the system are connected together in the model. The connections between the elements in the model define the order in which the model is simulated, as opposed to event and activity modeling where the order of simulation is defined by elements within the event or activity.

Event models are simulated by moving from event to event as dictated by the events themselves. There is no concern about any change between events, because none is allowed by the model. All events are equal in the sense that the scheduler knows no difference between an event that starts

an activity and an event that ends an activity. Any indefinite waits or conditional processing must be taken care of in the events by checking the status of the system. In a complicated model with many interactions between activities, the events can get quite complicated. It is often difficult to understand the flow of the model when event modeling is used since all events are separate and distinct. However, the event orientation is the most flexible discrete orientation.

In **activity models** each activity is checked at every time advance to see if the conditions for the start or stopping of an activity exist. This allows for very easy modeling of indefinite delays. In some cases the activity orientation is the easiest way to model a system. Like event models, activity models are disjoint. Since every activity is scanned at every time advance, activity models usually have long run times. The activity orientation is rarely used by itself, although it is used in part by many simulators [31].

Process models are simulated by following the order of the elements in the model. The simulation language will have elements to model definite and indefinite time delays and to model the flow of entities through a system. Process models are usually easier to build and understand than models built in the other discrete orientations. The relationship between activities is more obvious in process models. Process modeling lacks flexibility if one activity affects another

activity. The degree to which this is true depends upon the simulation language. Process oriented languages usually have aspects of the event and activity orientations included to make modeling easier.

A.1.3 Computer Languages for Simulation

There are many programming languages that can be used for writing simulation models. General purpose languages (such as BASIC, FORTRAN and PASCAL) can be used. There are general purpose simulation languages (such as GPSS, SLAM and ACSL) for each of the different modeling orientations. There are also special purpose simulation languages that are only useful for a limited range of simulations (material handling systems, integrated circuit design analysis, etc.).

An advantage of using a **general purpose language** is that a language can be chosen that the modeler and user already know. The modeler has the full flexibility of the general language at his disposal and is not constrained by a simulation language. The simulation is also very transportable because all that is needed to run it on another machine is a compiler or interpreter for that language. The main disadvantage of a general purpose language is that the modeler must take of all of the details of the model, such as writing integration routines, taking care of the advancement of time, etc. This can be quite tedious.

Using a **general purpose simulation language** solves the need for taking care of the tedious problems. Most simulation languages also add features that make writing most models much easier. Some examples are that they will keep track of the state of the system automatically so that it can be output later, plotting routines may be provided, and a set of instructions are provided that are specifically designed for simulation models. This makes modeling much easier. A draw back is that this new language must be learned by the modeler. General purpose simulation languages also restrict the modeler to the modeling orientation taken by the language. Models written in an orientation different from the language's orientation cannot be simulated with that language. The model will not be as transportable since it is harder to get a new version of the simulation language for a different machine than it is to get a compiler for a general purpose language for the machine. Some simulation languages are written in a general purpose language so if the source code of the simulation language is available, they can be ported relatively easily to a new machine.

A **special purpose simulation language** will be the easiest to use if you are only solving the specific types of problems the language was designed for. It would be very difficult, if not impossible, to use the language for any other types of simulation. The language is structured specifically for a particular application, so learning the

simulation language can be easier for someone who knows the application than learning a general purpose simulation language. Specialized output may also be provided. Special purpose languages will not be very transportable for the same reason that a general purpose simulation language is not very transportable.

A.2 Overview of SIMAN

This section provides an overview of the SIMAN simulation language. It describes the major subdivisions of the SIMAN language. The steps needed to run SIMAN are given. The essential terms needed to understand SIMAN are defined. The way a model is simulated is described [32].

SIMAN is a combined discrete-continuous general purpose simulation language, although the main emphasis of SIMAN is on discrete process orientation modeling. Discrete event orientation models and continuous models may be used by linking FORTRAN subroutines to SIMAN.

SIMAN is FORTRAN based, but for most discrete models the modeler or user does not need to know FORTRAN. However, by using FORTRAN the modeler has more flexibility in developing the model.

A.2.1 The SIMAN Framework

The SIMAN language is subdivided into five main user interaction points. The interaction points are called **frames**. The main two frames are the model frame and the experiment frame. The other frames are the output frame, the event frame and the continuous frame. SIMAN also has five processors to process models and output. They are the model, experiment, link, run and output processors.

The **model frame** contains the model statements. The model statements describe the system that is to be simulated. They do not specify under what conditions the model is to be simulated. The model frame is written in SIMAN statements. The statements form a block diagram of the model. There is an optional interactive program to input SIMAN models. This program uses graphics to draw the block diagram on the screen. This is completely compatible with the standard batch input mode where a text editor is used to input the model.

The **experiment frame** has SIMAN statements that describe how the model is to be simulated. The parameters that are to be used for a particular run are specified here. Run times, the number of runs, etc., are also specified. This must be entered in batch form.

The model and experiment are translated separately into what are called model and experiment files. The **model** and **experiment processors** are used to perform the translation.

The model and experiment files are then linked together by the **link processor** to form a run file. The separate translation allows changes in the experiment without having to retranslate the model. The model is exercised using the **run processor**. The run processor inputs and exercises the run file. The run processor can output a trace of the simulation, output a standard summary report, and can write simulation data to disk files.

The data written to files can be analyzed using the **output frame**. Commands for the output frame can either be entered in batch form or interactively when the output processor is running. There are commands in the output frame to plot the simulation data as a printer plot, to convert the simulation data to a form that can be read by non-SIMAN graphics programs, and to do statistical analysis on the simulation data. Data from more than one run and even from different models can be handled at once. This is useful for comparisons. The **output processor** uses the output frame commands directly to perform the output operations.

The **event frame** is coded in FORTRAN. The event frame can be used to add new commands to the model frame and it can be used to write event oriented models. There are many SIMAN subroutines that can be called from the events to take care of most of the housekeeping work in the events. The FORTRAN events must be linked to the SIMAN run processor using the link program that comes with FORTRAN. This forms a revised

run processor that includes the user written events. This revised run processor is used to simulate the run file that goes with the user defined events.

The **continuous frame** is also written in FORTRAN. The continuous state equations are coded using special SIMAN variables for the states and their derivatives. The user written continuous FORTRAN routine is linked to the SIMAN run processor. Even if no discrete model is used, the run file is needed to specify the run times and parameters for the continuous simulation.

A.2.2 Essential Terms

The SIMAN language uses some terms in a way that will be unfamiliar to most people that have never used a discrete computer simulation language. An understanding of some of these terms is essential in gaining an understanding of the SIMAN language. I will define these terms and explain how they are used in SIMAN. The terms defined in this section are entity, attribute, resource, queue and file.

The basic unit of data that causes everything to happen in SIMAN is called an **entity**. An entity could represent a customer in a bank model, a work piece in a manufacturing model, a piece of information to be processed in a computer model, or any number of other things depending upon the model. What an entity "is" in a model is defined by the

modeler. An entity is a general purpose "thing" made available by SIMAN to be used for whatever the modeler wants or needs. Not all entities in a model must represent the same object. Different objects may be included in the same model by defining entities that represent each of the objects. SIMAN does not know how an entity is defined. It is up to the modeler to assure that the model can distinguish between entities that represent different objects. This can be done by limiting entities that represent different objects to different parts of the model or by using some of the entities' attributes to define what object each entity "is."

Each entity has **attributes**. Attributes are a set of numbers that are associated with each entity. Each entity has a distinct set of attributes. Attributes are not shared, they are local to the entity. The attributes can represent anything the modeler wants them to. An attribute can be used to indicate what the entity "is," when the entity came into the system, what sequence it must take through the model, or anything else the modeler thinks might be useful. Attributes travel around the model with the entity they are associated with. In a sense, the entity <u>is</u> its attributes. There are as many sets of attributes in a simulation as there are entities. The attribute sets do not need to be the same size for all of the entities. The number of attributes is defined by the modeler for each entity object type.

The elements of system resources that are needed by an entity to perform a task are called **resources**. A resource could be a lathe, a space on an escalator, a fork lift, a cashier or whatever other element is shared by the objects (entities) in the modeled system. SIMAN has two types of resources: general purpose resources and material handling resources. The resources can be named by the modeler and can be referred to by name. This helps in understanding the model. The number of resources available can be changed from the model frame during the simulation. This allows the modeling of lunch breaks, change of shift, mechanical breakdowns, etc. The general resources are used to control the availability of the resources that are not used for movement of entities. The material handling resources are used to move objects around the system and model.

There are two types of material handlers. One is called a conveyor and the other a transporter. A conveyor is a continuous unit that can carry more than one entity at a time. A large entity takes more room on the conveyor than a small entity. Examples of a conveyor are a conveyor belt and an escalator. A transporter is a discrete device that can move more freely. Examples of a transporter are a fork lift, a robot and a porter in a hotel.

Before a resource can be used by an entity, the entity must gain control of the resource. If it cannot gain control of the resource it will usually wait for the resource to

become available in what is called a **queue**. At each point in a model when a resource is requested there must be a queue available for the entity to wait in if the resource is not available. All queues require a place to store the waiting entities. This place is called a file. A **file** is simply a place where entities are stored in a model. A file can be used for more general storage than just queuing.

A.2.3 Simulation Process Flow

This section describes how the entities move through the model during a simulation. I will leave the description of event and continuous processing until later when I describe those frames in detail. The simulation is entity (data) driven. Each entity moves through the model independently of every other entity in the model except that an entity may have to wait to use a shared resource or one entity may be required by the model to wait for the arrival of another entity at a certain point in the model. The entities are processed in parallel.

The processing starts when an entity gains "control" of the simulation. The entity then moves through the model, being affected by the model statements, until the entity is destroyed or is delayed by a model statement.

After processing is done on one entity, the list of scheduled entity processings (called the task list) is

checked. The simulation time is set to the next processing time on the list. The entity that is scheduled to be processed next is then processed. The entity starts from the point in the model where it was delayed and is processed until it is delayed again. If two entities are scheduled to be processed at the same time, the one that was put on the task list first is processed first.

A delay can either be definite or indefinite. If the delay is definite, the entity is scheduled to regain control of the simulation after the delay. For indefinite delays the entity is waiting for the system state to reach a certain point, such as a resource being available. The entity is processed at the instant when the state changes to end the delay.

No time delays can occur during the processing of an entity; the processing occurs in an instant of time. The entity cannot affect the state of the system while it is waiting for its next processing time. Changes can occur only at discrete instants in time, hence the name discrete modeling.

Parallelism is achieved by processing an entity while every other entity is being delayed. Each entity moves through the model until it is destroyed (leaves the system), or the run ends. This relatively simple scheme allows for very complex models with many things going on at the same time.

A.2.4 SIMAN Variables

SIMAN has many variables built in that make the modeling task much easier. The variables consist of user assignable variables, SIMAN system variables, SIMAN random variables, table functions and user coded functions.

The user assignable variables are variables available to the modeler to define as needed. See Figure A.1 on page 110 for a list of the variables and a description of what they can be used for.

The SIMAN system variables are used to tell the system status. These variables are set by the simulator. A complete list is given in Figure A.2 on page 111.

SIMAN provides many random variables for a modeler's use. The modeler can choose which of ten random number streams to use for each function call. The modeler also must specify from which parameter set the needed data for the random variable is to be taken. The values needed specify the specific shape of the random variable. For example, for a normal distribution, the mean and standard deviation must be specified. The complete list of random variables is given in Figure A.3 on page 112.

A set of user definable table look-up functions is provided for the modeler. A table is defined in the experiment frame. Table values are defined at even increments of the independent variable. At simulation time, the value

Variable	Description
X(I)	Real valued global variable - available for general use.
P(I,K)	Real valued parameter K in parameter set I - Defined in the experiment frame, definition used by the SIMAN random variables and to allow changes in model parameters without changing the model.
J	Integer valued index variable - used for general indexing, also used to return a pointer after several model commands.
S(I)	Real valued continuous state variable I - used to define the state equations in the continuous frame, can be referenced in the model and event frames, a change will cause a step change in the continuous system.
D(I)	Real valued derivative of S(I) - used to define the state equations in the continuous frame, can be referenced and changed in the model and event frames, integrated by SIMAN to obtain S(I).

Figure A.1 SIMAN User Assignable Variables [33]

Variable	Description
TNOW	The current simulation time.
NE(I)	The number of entities on route to station I.
NQ(I)	The number of entities in queue I.
NC(I)	The current count for counter I.
NR(I)	The number of busy units of resource I.
MR(I)	The total number of units of resource I.
NT(I)	The number of busy units of transporter I.
MT(I)	The total number of units of transporter I.
LC(I)	The length of occupied cells on conveyor I.

Figure A.2 SIMAN System Variables [33]

Name	Distribution Type (Required Parameters)
BE	Beta distribution. (Theta, Phi)
CO	A constant from the parameter set. (Constant)
DP	A discrete probability distribution. The user defines the cumulative probabilities that the random variable will return a certain value.
ED	The random variable to be used is specified in the experiment frame.
ER	Erlang distribution. (Exponential mean, K)
EX	Exponential distribution. (Mean)
GA	Gamma distribution. (Beta, Alpha)
NP	Poisson distribution. (Mean)
RA	Uniform random sample in the interval 0 to 1.
RL	Lognormal distribution. (Mean, Standard deviation)
RN	Normal distribution. (Mean, Standard deviation)
TR	Triangular distribution. (Minimum, Mode, Maximum)
UN	Uniform distribution. (Minimum, Maximum)
WE	Weibull distribution. (Beta, Alpha)

Figure A.3 SIMAN Random Variables [34]

returned from the function is linearly interpolated from the values in the table.

The modeler can also program, in FORTRAN, user defined functions. The function can use anything available in FORTRAN, such as a READ statement, or anything available from SIMAN, such as the system status.

A.3 The Model Frame

The model frame is what the modeler will spend the most time developing. A model may be written directly in SIMAN or it may be written in some other form and converted to SIMAN statements. It is fairly easy to convert from most representations of a system to a block diagram model.

The model frame contains the SIMAN block diagram model of the system. The block diagram consists of block labels, block functions, block modifiers and comments. Block labels are used to identify a particular block in a model. A label is made up of alphanumeric characters, including blanks, and may be up to eight characters long. The block functions are the blocks that make up most of the model. They will be described below. A block modifier specifies special processing for that block. See Figure A.4 on page 114 for a description of the modifiers. Comments are used by the modeler to make understanding the model easier.

Modifier	Description
DETACH	Do not connect this block to the next block. Only used by QUEUE blocks. Needed when the QUEUE is associated with a QPICK or MATCH block. Also used to use the QUEUE block for entity storage.
DISPOSE	Destroy the entity after processing at the block.
MARK(n)	"Mark" (store) the arrival time to the block in attribute n.
NEXT(label)	Go to label after processing at the block.

Figure A.4 SIMAN Block Modifiers [35]

SIMAN has ten basic block types. Each block type is a grouping of one or more block functions. Three of the block types have more than one block function. The other seven block types uniquely specify a block function. The block types are shown in Figure A.5 on page 116. For the block types that are also block functions, the parameters that can be specified in the block function are listed. Figure A.6 on page 117, Figure A.9 on page 120 and Figure A.10 on page 121 list all of the block functions associated with the block types operation, transfer and hold, respectively. The parameters that can be specified in the block are listed.

Many of the block function parameters have default values, so the modeler does not need to specify all of the parameters. A name is an up to eight character alphanumeric user defined label for resources and material handling elements. Almost all of the numeric parameters can be specified as an integer, as the index variable J plus-or-minus an integer, as the station number plus-or-minus an integer, as an attribute of the entity or as one of the global variables. When an expression can be specified, as for the ASSIGN block and any time delay, an algebraic combination of any of the variables listed in "SIMAN Variables" on page 109 can be used. This allows time delays and other values to be based on the random variables and the system status. When a condition is allowed, a combination of expressions and logical operators can be used.

Block Type	Description (Parameters)
OPERATION	Used to model a variety of processes. See Figure A.6 on page 117 for a complete list of functions.
HOLD	Used to model situations where an entity may be delayed based on system status. Must be preceded by a queuing facility to hold the entity if it is delayed. See Figure A.10 on page 121 for the specific blocks.
MATCH	Delays entities in a set of queues until each queue contains an entity with the same value of a specified attribute. Each entity then moves on independently of each other. (Attribute number to compare, labels of QUEUE blocks, labels of blocks entities move to after a match)
QUEUE	Provides waiting space for entities that are delayed at hold or match blocks. (File number, queue capacity, label of where entities go when the queue is full)
BRANCH	Allows conditional, probabilistic and deterministic rerouting of entities in a model. An entity may take more than one branch, in which case, an identical copy is made of the entity. (Maximum number of branches that can be taken, branch condition, label to branch to)
PICKQ	Used to select from a set of following QUEUE blocks. (Rule on how to select a queue, label of each QUEUE, label of where the entity goes when all of the queues are full)
SELECT	Used to select between a set of following hold blocks. (Rule on how to select which hold block, labels of the hold blocks)
QPICK	Used to select from a set of preceding QUEUE blocks. (Rule for selecting a queue, label of each QUEUE)
TRANSFER	Used to model transfers between stations via material handling systems. Figure A.9 on page 120 has a complete list.
STATION	Defines the point in the model where an entity is moved when a transfer block is used. (Station number(s) represented by this block)

Figure A.5 SIMAN Block Types [36]

Block Function	Description (Parameters)
General Functions	
ASSIGN	Make a general assignment. (Variable = expression)
CREATE	Cause a batch of entities to arrive at the system. They move on from this block. Also specify when the next entities are to be created at this block. (Number in batch, time between creations, maximum number of batches that can be created)
DELAY	Delay the entity a specified time. (Delay time)
DETECT	Detect when a continuous state variable or derivative crosses a specified value - which may be another continuous variable or a constant. When the crossing is detected an entity is created at this block. (Variable to be checked, crossing direction, value that must be crossed, how close the crossing time must be calculated by SIMAN, station numbers for which this applies)
EVENT	Invoke a user written event. (Event number)
FINDJ	Find the value of the index J meeting a specified condition. J is set by this function. (Search condition as a function of J, range of J over which the search is made)
SIGNAL	Send a signal. Releases entities at a WAIT block. (Signal code number)
SPLIT	Split a previously formed group into its individual members. (No parameters)

Figure A.6 Operation Block Type Functions (Page 1 of 3) [37]

Block Function	Description (Parameters)
Resource Functions	
RELEASE	Release units of the specified resource. The resource is now available for other entities to use. (Resource name, number of units to be released)
ALTER	Change the capacity of the specified resource. (Resource name, capacity change)
Material Handling Functions	
FREE	Exit a transporter. The transporter is now free to carry other entities. (Transporter name)
ACTIVATE	Set the status of a transporter to active. The transporter can now move. (Transporter name)
HALT	Set the status of a transporter to inactive. The transporter cannot move. (Transporter name)
EXIT	Exit a conveyor. (Conveyor name, number of cells released)
START	Set the status of a conveyor to active. The conveyor starts moving. (Conveyor name)
STOP	Set the status of a conveyor to inactive. The conveyor stops moving. (Conveyor name)
Statistics Functions	
COUNT	Increment the specified counter. (Counter number, increment)
TALLY	Record an observation value. (Tally number, value to be recorded)

Figure A.7 Operation Block Type Functions (Continued, Page 2 of 3) [37]

Block Function	Description (Parameters)
File Functions	
SEARCH	Search a file for an entity meeting a specified condition. After the search, the index J points to an entity meeting the condition. (Condition as a function of J, file to be searched, range of J over which the search is made)
COPY	Make an identical copy of an entity in a file. The new copy is sent to the labeled statement specified. (File and position in the file of the entity to be copied, label where copy is sent)
REMOVE	Remove the specified entity from a file and send it to the desired label. (File and position in the file of the removed entity, label to where the entity is sent)

Figure A.8 Operation Block Type Functions (Continued, Page 3 of 3) [37]

Block Function	Description (Parameters)
ROUTE	Move the entity to the specified station. No material handling resource is needed. The transfer time is specified in the block. (Transfer time, destination station number)
CONVEY	Use a conveyor to move the entity to the specified station. The transfer time is determined by the distance along the conveyor between the stations and the speed of the conveyor. (Conveyor name, destination station number)
TRANSPORT	Use a transporter to move the entity to the specified station. The transfer time is determined by the distance between the stations and the transporter's speed. (Transporter name, destination station number)

Figure A.9 Transfer Block Type Functions [38]

Block Function	Description (Parameters)
Resource Functions	
SEIZE	Wait until the required number of units of a resource are available. Allot the resources to the entity. (Name and number of units of the resource, priority relative to other SEIZE blocks)
PREEMPT	Wait until a unit of the requested resource is allotted to the entity. The entity may preempt a resource currently being used at a lower priority. (Resource name, preempt priority, label to which preempted entity is to go, attribute of preempted entity in which the remaining delay time is stored)
Material Handling Functions	
ACCESS	Wait until a specified number of consecutive conveyor cells are available at the accessing station. Allot the cells to the entity. (Conveyor name, number of cells required)
REQUEST	Wait until a transporter is allotted to the entity and the transporter arrives at the requesting station. (Transporter name, priority relative to other REQUEST blocks)
Condition Functions	
WAIT	Wait until the specified signal is sent from a SIGNAL block. (The signal code number)
SCAN	Wait until a specified general condition is met. This is used for activity orientation modeling. (Scan condition, the range of stations the entity must be at to enable the scan)

Figure A.10 Hold Block Type Functions (Page 1 of 2) [38]

Block Function	Description (Parameters)
Set Functions	
COMBINE	Wait until the specified number of entities are in the preceding queue. Combine the entities into a permanent set and create a representative of the set. The original entities are destroyed. (Set size, how the attributes of the new entity are to be determined)
GROUP	Wait until the specified number of entities are in the preceding queue. Group the entities into a temporary set and create a representative of the set. The original entities are saved and can be recovered using the SPLIT block. (Set size, how the attributes of the new entity are to be determined)

Figure A.11 Hold Block Type Functions (Continued, Page 2 of 2) [38]

One of the most common sequences of blocks is QUEUE, SEIZE, process, and RELEASE. This sequence allows an entity to gain control of a resource, perform any process that the resource does on the entity (usually modeled by DELAY), and then frees up the resource for another entity to use. An example is an inspection station. The part would request the inspector, it would wait in the queue if necessary, then, when the inspector was available, the inspector would inspect the part (this is modeled by a delay and assignment of an attribute indicating pass or fail), and then the inspector would be free to inspect another part. Both the time delay and the assignment of pass or fail could be based on random distributions.

The QUEUE, SEIZE, process, RELEASE sequence can be generalized as the sequence: wait in a queuing facility until the system status is as needed, perform a process and release any resources used. The queuing facility will be at least one QUEUE block and could include a QPICK which allows more than one queue to form in front of a hold block. The check on system status will be done at a hold or match block. These include waits until a resource or material handler is available, to waiting for other entities to arrive, to a general status check. A SELECT block allows several hold blocks behind one QUEUE or QPICK. The processing can be any block function, including more queues and holds. This

structure is very flexible and allows a wide variety of modeling options.

Most ways of prioritizing who gets a resource first and who can "steal" the resource from whom can be modeled using the SEIZE and PREEMPT blocks. A PREEMPT block is always a higher priority than a SEIZE block.

An entity may be rerouted using a BRANCH block, a selection rule block or a transfer block. The BRANCH block allows an entity to be branched based upon a conditional statement (branch if a is less than b), or on a probability independent of the system status (branch 20% of the time), or it can be branched deterministicaly (branch if no other branch has been taken). All of these may be combined in one BRANCH block, and the block may have as many branches as needed. If more than one branch could be taken, copies of the entity are made and sent to each branch. The maximum number of branches that can be taken is specified by the modeler. The usual case is one branch taken. There is no case for the entity to "fall through" the block. At least one branch must be taken or the entity will be destroyed. The branch statement is very flexible and fairly easy to use.

The second type of rerouting involves selection rules. The PICKQ, SELECT and QPICK blocks use selection rules. A queue or hold block is chosen based on the selection rule. These include picking the block to go to based on the shortest or longest queue, or the most or least resources

available, or they can be chosen randomly or in several other ways which are listed in reference [39].

The last way to reroute an entity is using a transfer block. (See Figure A.9 on page 120) The entity moves to the STATION block specified in the transfer block. To use the transfer blocks CONVEY and TRANSPORT, the material handler must be requested at a hold block first. For CONVEY and TRANSPORT, the entity's station number is important in calculating when the material handler arrives at the station and how long the transfer will take. A special integer attribute of each entity, called the **M attribute** or the **station number**, is used to tell what station the entity is at. The M attribute is set when a entity arrives at a STATION block. It can also be changed by the modeler using an ASSIGN block. The distances between stations and the speed of the material handler is specified in the experiment frame. Unfortunately, the speed of the material handler cannot be varied during the model run. This makes it impossible to use the material handling functions to model the effects of loads on the material handling system.

SIMAN has special indexing features that allow a modeler to create what are called macros. A macro is a part of the model that represents more than one station. The processing at each of the stations must be similar for the macro to be useful. In a macro all of the queues and resources must be indexed using the entity's M attribute so that the queues and

resources of the different stations represented by the same block diagram can be kept separate. The macro capability is very useful when there are parts of the system that are identical or almost identical to another part of the system. An example is a job shop model. Two lathes could be modeled by the same set of blocks. It is possible that all of the machines in the shop could be defined by the same set of blocks if their processing is similar enough to each other. The model would then only require three station macros, the arrival station, the machine stations and the exit station.

The TALLY block is used to record data for output. A common value recorded is the in-system-time. This is the time the entity is in the system. To find this time the MARK block modifier is used on the CREATE block. The last block the entity goes to is a TALLY block recording the interval of time between the mark time and the current time. The TALLY block will have the DISPOSE modifier to destroy the entity. The TALLY block can record any expression.

The model frame can be entered in either interactive or batch form. The two input methods are completely compatible and may be used interchangeably. This is necessary because the interactive input requires a graphics terminal to input the model. A graphics terminal is not always available, so the batch input may sometimes have to be used. Also, it is much easier to get non-graphics printouts than to get graphics printouts.

The interactive input mode allows the modeler to input the blocks from a menu. The block diagram is displayed graphically on the terminal as it is built. Block parameters are typed on the keyboard. Labels are inserted to the left of the block. Comments can be added to the right of the block. Block modifiers are appended to the bottom or right side of the block. When using interactive input, the modeler is told of errors as they are typed in. The interactive input program does not replace the model processor. This would be a nice feature because one step could be saved in running a simulation. The interactive checking for errors still speeds things up.

In batch input a standard text editor is used to enter the input commands. The first nine columns are used for labels. The rest of the line is used for the block, modifiers and comments. The labels, blocks and modifiers must be entered in upper case. Comments, which can be put on the same line as a block, can be entered in mixed case. If more than one line is needed for a block, the block is simply continued on the next line, as long as the end of the first line indicates that there is more to the block. This is done by ending the line with a comma or a colon, the two field separators used by SIMAN. For some of the blocks the point where a comma is required instead of a colon seems inconsistent with the comma and colon usage for the rest of the blocks. The modeler is not allowed to put comments on the

line that is to be continued. This means that the blocks that need comments the most, the long complicated ones, are very hard to comment. A block modifier is placed after the block using a colon as a separator. The end of the block and modifier is indicated with a semicolon. Anything on the line after the semicolon is considered a comment and is ignored by SIMAN. The frame must start with a BEGIN; statement and end with an END; statement.

After the model frame has been entered, it must be translated using the model processor. This converts the user file to a form that can be read by the rest of SIMAN. It also checks for syntax errors in the model input. Any error messages are printed on the line after the error. No total number of errors is given at the end of the translation. For a long model this is very annoying because the entire listing must be checked to see if there were any errors. The model processor translates about two hundred lines (a moderate sized model) in two minutes on an IBM PC.

A.4 The Experiment Frame

The experiment frame is where the data to drive the model is stored. The model can easily be set up so that any value that may need to be changed in the model is placed in the experiment frame. This makes it so the model never has to be changed once it is working. Many different experiments

can be run on the same model - all by just changing the experiment frame and rerunning the model.

The experiment commands, called elements, for the most part can be entered in any order. There are several elements that must be put in the beginning of the frame if they are used. SIMAN does not allow an element to appear more than once in the experiment frame. If there is more than one set of data to be specified by an element, the additional specifications follow the first to form one long statement. The statement can be spread out over as many lines as needed.

Unlike the model frame, in the experiment frame variables may not be used to specify values. Only real or integer numbers may be used. This usually presents no problems since the values in the experiment frame generally do not need to depend on the status of the system. The values in the parameter set (which is defined in the experiment) may be changed from the model during a run if changes are needed. There are many defaults allowed in the experiment frame, making coding easier. All of the experiment elements are listed by function type in Figure A.12 on page 130, Figure A.13 on page 131, Figure A.14 on page 132, Figure A.15 on page 133, and Figure A.16 on page 134.

The only restrictions on the ordering of the experiment frame is that the PROJECT element, if used, must be the first element in the frame. The DISCRETE and CONTINUOUS elements can only be preceded by the PROJECT element or each other.

Element	Description (Parameters)
PROJECT	Generates a summary report. (Project title, analyst name, date)
DISCRETE	Defines the maximum number of several storage dependent variables used in discrete models. Required when a discrete model is used. (Maximum number of: concurrent entities, attributes per entity, files, stations)
CONTINUOUS	Required when a continuous model is used. Defines the integration variables, tells SIMAN the setup of the equations to be solved, and how often continuous variables are to be recorded. (How often continuous variables are to be saved, number of differential and state equations used, minimum and maximum allowable step size, maximum allowable integration error, what to do if step size smaller than the one specified is needed to keep within accuracy)
REPLICATE	Required by all models. Defines the number of runs, the length of the runs and whether or not to reinitialize between runs. (Number of runs, beginning time of first run, maximum length of each run, whether or not to initialize the system status or save observations between runs)

Figure A.12 SIMAN Overall Defining Experiment Elements [40]

Element	Description (Parameters)
DISTANCES	Use with transporters. Defines distance sets. (Distance set number, range of stations included in set, distance between stations)
SEGMENTS	Use with conveyors. Defines segment sets. (Segment set number, station segment set starts at, station/distance pair defining the next segment in the set)
PARAMETERS	Define parameter sets. (Parameter set number, parameter values)
DISTRIBUTIONS	Define the random distribution ED(I). (Distributions number, distribution to be used)
RULES	Specifies the selection rule ER(I). (Rules number, selection rule to be used)
TABLES	Defines a table function. (Table number, low value and increment of the independent variable, dependent values)
RANKINGS	Change the default queue ordering. (Range of files to be redefined, ranking criterion)
EVENTS	Cause an event to be called when a continuous variable reaches a certain value. (Event number, variable to be checked, variable to be crossed, crossing direction, tolerance required)

Figure A.13 SIMAN Parameter Definition Experiment Elements [40]

Element	Description (Parameters)
RESOURCES	Defines any resources used. (Resource number, name and capacity)
TRANSPORTERS	Defines any transporters used. (Transporter number, name, capacity, velocity, initial position and status, distance set to use)
CONVEYORS	Defines any conveyors used. (Conveyor number, name, velocity, initial status, length of each cell, segment set to use)

Figure A.14 SIMAN Resource Definition Experiment Elements [40]

Element	Description (Parameters)
ARRIVALS	Create batches of entities to arrive at the specified location in the model. (Arrival number, location entities enter the model, time of creation, batch size, the entities' attributes)
INITIALIZE	Initialize variables at the start of a run. (Variable to be initialized, value it is to be initialized to)
SEEDS	Initialize the random number streams. (Stream number, seed value, whether or not to reinitialize between runs)

Figure A.15 SIMAN Initialization Experiment Elements [40]

Element	Description (Parameters)
COUNTERS	Defines any counters used. Causes output on the summary report. (Counter number, maximum value counter can take, identifier printed on summary report)
TALLIES	Required to identify all TALLY blocks in the model. Causes output on the summary report and/or to disk. (Tally number, summary report identifier, disk file data is to be written to)
DSTAT	Causes all of the changes in a discrete SIMAN variable to be recorded. (DSTAT number, variable to be recorded, summary report identifier, disk file data is to be written to)
CSTAT	Causes continuous variables to be recorded at the interval specified in the CONTINUOUS element. (CSTAT number, variable to be recorded, summary report identifier, disk file data is to be written to)
OUTPUT	Output the value of a variable at the end of each run. This is usually a statistical variable. (Output number, variable to be written, disk file data is to be written to)
TRACE	Cause a trace to be printed when the specified condition is true. Print the specified variables. (Start and stop times for the trace, trace condition, up to four variables to be printed each trace time)

Figure A.16 SIMAN Output Experiment Elements [40]

The PROJECT element causes a summary report to be printed and allows the report to be named and dated. The summary report prints data pertinent to the model at the end of each run. The contents of the summary report are determined by other experiment elements. The DISCRETE element is required when a discrete model is to be simulated. The CONTINUOUS element is required when a continuous model is used. Other than these three elements the rest of the elements in the experiment frame can be in any order. The REPLICATE element is required, but can be placed anywhere in the experiment. This element is used to control the simulation.

The resource definition elements, Figure A.14 on page 132, give the resources names that can be used in the model. They also define what limits and characteristics the resources have. The capacity is the number of identical units of that resource, not how many entities can be handled by one resource.

The DISTANCES and SEGMENTS elements are used by transporters and conveyors, respectively, to find how far the different stations are from each other.

One of the most powerful elements in the experiment is the PARAMETERS element. This element is a two dimensional array of values. The first dimension defines the parameter set. The second dimension defines the parameter set values. The modeler can use these parameters in any way desired. This allows the modeler to put all values that may change between

experiments in the parameter set. Then the user only needs to change the parameter to change the value in the model. The user never needs to modify the model. The parameters can even be used to specify conditional processing of model blocks. This allows minor changes in the model to be simulated without changing the model. The parameters are also used by the SIMAN random variables. The modeler specifies which parameter set each call to a random variable should get its data from. The random variable then goes to the parameter set and gets the parameters it needs. There may be as many parameter sets as needed and each set may have any number of parameters. Different parameter sets may have different numbers of parameters.

The DISTRIBUTIONS and RULES elements allow the user to define what random variable or selection rule is to be used in the model. This, again, allows the modeler to construct a model that does not need to be modified to run very many, very different experiments.

The output elements, Figure A.16 on page 134, cause output to the screen, disk or both. COUNTERS, TALLIES, DSTAT and CSTAT cause output on the summary report. The user can specify an identifier to be printed on the summary report. The identifier can be up to twenty characters including blanks. All variables output on the summary report must be specified by one of these elements. There are no default variables printed on the summary report. This keeps the

summary report from being cluttered with unnecessary output. These elements, except COUNTERS, can also be used to output run data to disk. This is an option, so only the data that the user wants written is sent to disk. Any data that is sent to disk must have an entry on the summary report even if the entry has no meaning.

The TRACE element causes a trace of the run to be performed. This is very helpful for debugging the model. The trace is output on the screen. A trace condition can be specified so that only certain steps are traced. By using the trace condition only certain events are traced, allowing the user to follow the simulation run, but not having to see every step. The types of output obtainable from SIMAN are described in more detail in "Output Available" on page 139.

The experiment frame must be entered in batch mode. The elements are typed in using a text editor. The first statement must be BEGIN; and the last must be END;. The experiment may start in any column. The elements must be entered in uppercase. The lines are continued as in the model frame, except I have had some trouble splitting a line after a comma. Comments are allowed as in the model frame. The problem with not being able to comment a line that is continued is even more of a problem in the experiment than in the model. The experiment has very long lines. If the data is not commented, the user will have no idea what each piece of data is, and will not be able to run experiments.

After the experiment is entered, it is translated into the experiment file by the experiment processor. Any syntax errors are printed. On an IBM PC, this takes about a minute on a one hundred line experiment, which is about as large as an experiment will get. The run file and experiment file are linked to form the model file by the link processor. Additional errors may be found in this step. There is no provision for linking more than one model file to an experiment. This forces the modeler to store the model frame in one file and translate the whole thing when any change is made. On the PC, the link step takes over a minute for a moderately sized model.

After the run file has been created it may be simulated using the run processor. When the run processor is invoked a banner message with the license number is displayed. The program waits for an input from the user to make sure the banner is read. The run file is then loaded and executed. Trace output is printed as the program executes. At the end of the run a summary report is printed to the screen. The trace and summary report is repeated for each of the runs specified in the experiment. After the last, run the program stops. The time needed to load the run processor and a moderately sized model is about 30 seconds. The run time will depend greatly upon the model. It takes a second to simulate about 50 changes in a discrete model.

A.5 Output Available

SIMAN can generate many different types of output. I have already mentioned the trace and summary reports in "The Experiment Frame" on page 128. SIMAN also has an output processor that can make printer plots and convert the SIMAN output data to a format that can be used by other plotting routines. The modeler can also specify output by using FORTRAN write statements from within a discrete event. The types of output can be split into run time output and the output frame.

A.5.1 Run Time Output

The run time output, except for FORTRAN write statements, is controlled by experiment frame elements. See Figure A.16 on page 134 for a list of the output elements. The summary report output lists the identifier, the average value of the variable, the standard deviation, minimum and maximum value, and the length of time the observations were made over. The output is grouped into sections defined by the experiment elements. If data is written to disk the observation time and value of the variable is output every time the variable changes. The data from successive runs are output to different disk files.

The modeler can create a specialized trace for a model by coding events that print the status of the system. The events are placed in the model frame at critical points so it is easy to keep track of the model's progress. Different events can be used for different points in the model.

Events also can be used to generate a specialized summary report. The standard summary report can be suppressed by not including the PROJECT element in the experiment.

A.5.2 The Output Frame

The output frame is used to process data written to disk during one or more simulation runs. There are three basic types of elements in the output frame: plotting, conversion of data for other software and statistical analysis elements. All of the elements are listed in Figure A.17 on page 141.

The plotting elements draw plots on the screen. Special graphics are not used, so the plots can be displayed on any terminal or printer. The plotting resolution is very low. The user can specify the parameters and time range for the drawings. Tables can also be generated.

The DIF and EZPREP elements convert data from the internal SIMAN binary form to a format that can be easily read by an external graphics program. EZPREP outputs the data to files in a FORTRAN E12.5 format. One value is written per line, and only one variable is written per file. This data

Element	Description
Plotting Elements	
BARCHART	Draw a bar chart.
CORRELOGRAM	Draw a correlogram.
HISTOGRAM	Draw a histogram.
PLOT	Draw a plot of time persistent variables.
TABLE	Generate a table of values and display it on the screen.
Conversion of Data for Other Software Elements	
DIF	Converts the run data into the DIF format.
EZPREP	Converts the run data into columns of data in FORTRAN E12.5 format.
Statistical Analysis Elements	
FILTER	Generates a filtered data set using batching and truncation.
INTERVALS	Constructs confidence intervals.

Figure A.17 SIMAN Output Frame Elements [41]

is in ASCII format and can be easily read by most programs. The data is specifically set up to allow direct input into the Tektronix PLOT10 Easy Graphing Software. The DIF element converts data to the DIF format, which is a standard used by many personal computer software packages, including Lotus 1-2-3 and VisiPlot. The converted data can not only be used as input to a plotting program, it can also be used as input to almost any post processor, including statistical analysis packages.

The FILTER element will remove the transient portion of the data at the beginning of a run and will put the rest of the data in batches so that for a non-terminating system, the result will be observations that are approximately stationary, independent and normally distributed [42].

The INTERVALS element generates confidence intervals for a specified set of observation variables.

The output frame can be entered interactively when the output processor is running or it can be entered in batch form. Either way, the frame is entered in the same format. There are no menus in the output processor. The output frame must be entered in upper case. The input format is like the experiment frame. An element may be repeated as often as needed. The output processor uses the output frame elements directly, so no translation is needed. The elements are executed as they are entered, so further output commands can be based on the current output results.

When using interactive input, the processor is very unforgiving of typing errors. One mistake in the element will force the user to type the entire element over again. Some of the elements can get quite long and detailed, covering several lines. It often takes many tries to get the input right. A menu driven input routine would aid considerably.

A.6 The Event Frame

FORTRAN subroutines can be used for many things in SIMAN. They can be used to define the user function (UF) and user rule (UR), to initialize and wrap-up runs, to code continuous models, to create new SIMAN block functions, and to model using the event orientation. Their use in continuous models is described in "The Continuous Frame" on page 146.

SIMAN calls the subroutine PRIME at the beginning of each run and the subroutine WRAPUP at the end of each run. A null routine is provided by SIMAN so the modeler does not have to write these routines if they are not needed. The modeler can do any initialization needed and any wrap-up using these routines. The PRIME routine is especially useful for initializing variables used by any modeler written FORTRAN routines. The WRAPUP routine is very useful for printing out a specialized summary report.

The event frame is simulated differently than the model frame. Instead of being controlled by the entity, the event

is controlled by the program flow of the FORTRAN routine. The routine affects the entity and the system status. Since the entity is not controlling the routine, it is easy to manipulate more than one entity at a time from an event. The drawback to this is that the event is responsible for moving the entity around the model and for scheduling when the next event should occur. This is taken care of automatically in the model frame.

A complete model could be created using the event modeling orientation and SIMAN events. Since it is generally easier, quicker and more understandable to use the process orientation, the modeler would rarely want to do much work using the event orientation. But the event orientation is more flexible than the process orientation, so it must be used sometimes. SIMAN has a very easy interface between the event frame and the model frame. The EVENT block moves the entity from the model frame to the event frame. To get from the event frame to the model frame, an event may schedule an entity to arrive at a station or queue in the model frame.

One of the largest advantages to using the event frame is the ease in which several entities can be manipulated at once. Also, complex math is much easier in FORTRAN than in the model frame.

The event frame can be used to create a new model function. This is especially useful when the mechanics of the function are not important in understanding the model and

would only clutter the model frame. The function can be "hidden" in the event frame. The new function is invoked by the EVENT: N; block in the model frame. When an entity enters this block the modeler written subroutine EVENT(L, N) is called, where L is a pointer to the calling entity and N is the event number. The EVENT subroutine can then call the subroutine that processes event N. This subroutine can perform the new function desired. If the event does not put the entity in a file or send it to some other part of the model, the entity moves on to the block after the EVENT block when the event processing is done.

All of the SIMAN variables can be read in FORTRAN routines. Many functions and subroutines are provided by SIMAN that allow the modeler to manipulate the system. Functions are provided to perform file manipulations that are impossible to do in the model frame. Also, all of the statistical variables, such as average and standard deviation, are available during the run. There is one routine, LRANK(NR, IFL), that is listed as available, but it is never described in the manual. I have found a subroutine and a function that are never mentioned in the manual. They are the subroutine SIGNAL(NSIG) which sends signal code NSIG, and the function P(I, J) which returns the value of parameter J of parameter set I.

Unfortunately, SIMAN does not allow an entity to gain the use of resources from within an event. Entities cannot

be formed into sets or a set split from an event. This restricts the power of the event frame.

SIMAN allows much flexibility in how a model is constructed. Almost all discrete models can be done totally in the model frame using the process orientation. All models can be done using the event orientation. Most models can be done as a combination of the two. This allows the modeler to choose the way he feels is best for each model. He is not forced into one way by SIMAN.

The event frame is coded in standard FORTRAN. After the events are compiled, they must be linked to all of the SIMAN routines. The linker that comes with FORTRAN is used for this. This is a very slow process on an IBM PC. A link takes over six minutes. This time discourages the use of the event frame. The result of the link is a revised run processor that is used to input and simulate the run file.

A.7 The Continuous Frame

SIMAN allows continuous models to be simulated. The ability to simulate continuous models is intended primarily as an add-on to discrete models. The continuous models can be run by themselves. A model frame is still required, but it can be just BEGIN; and END; statements. The continuous frame is written as state and differential equations. In state equations a state of the system is represented as a

mathematical combination of states, states' derivatives and system variables. Differential equations represent the derivative of the state as a combination of states, states' derivatives and system variables. Both state and derivative equations can be used in the same model.

The model is coded in the FORTRAN subroutine STATE. The modeler uses the SIMAN global variables S(I) and D(I) to code the state and differential equations. S(I) represents the state I and D(I) represents the derivative of the state I. The modeler must order the equations so that any variables that are needed by a state or differential equation is computed before the equation it is used in is computed. The modeler can use any feature of FORTRAN to return the states and derivatives. This allows the modeler to model nonlinearities, such as step inputs. Also, a routine could be written that converts a frequency domain input into the time domain values needed by SIMAN.

When the model is simulated, SIMAN uses the Runge-Kutta-Fehlberg fourth-fifth order integration algorithm to solve the differential equations [43]. The algorithm uses a variable integration step size. This allows the user to specify what accuracy is desired. The step size is adjusted to give the desired accuracy with the shortest computation time. Other integration methods may be used, but they must be coded by the modeler as difference equations in the subroutine STATE.

Discrete models can directly manipulate the continuous states and derivatives to force a change in the continuous model.

The continuous model can affect discrete models in several ways. Discrete models can base processing on the continuous model's states and derivatives. The modeler can also use a DETECT block or an EVENTS experiment element to detect when the status of the continuous model reaches a predetermined point. (See Figure A.6 on page 117 and Figure A.13 on page 131). When the continuous status reaches the desired point, an entity is created and moves on from the DETECT block or is processed by the specified event.

To run the continuous model, the subroutine STATE must be compiled and then linked to the SIMAN subroutines in the same way as, and with any, discrete events. The run times depend on the complexity of the model. On an IBM PC, a second order system which modeled a sine wave needed four seconds to simulate a cycle.

A.8 Evaluation and Conclusion

In this section I will provide an evaluation of the SIMAN language as a whole. I will discuss the usefulness and ease of use of the language. I will finish with an overall conclusion on the usefulness of SIMAN.

A.8.1 Evaluation of SIMAN

The SIMAN language is primarily process orientated. Models in this orientation are fairly easy to understand. It is straight forward to code SIMAN from other algorithmic representations of the system. One problem with the process orientation in general, is that it is difficult to have two entities affect each other. SIMAN has features that make it easier to affect one entity with another. In the event orientation, there is no difficulty with entities affecting each other. So, that orientation could be used for the parts of the model where entities affect each other.

The ability to add events is a very powerful feature. Even though most models can be written without using events, events can be used for clarity. The details of a process that is not that important to the model can be put in an event to form a new function. By using these new functions, SIMAN can be adapted for specialized modeling easily. The modeler is given the choice, in most cases, whether it would be better to use the event or model frame to model any part of the system.

Unfortunately, SIMAN does not support all of the model frame elements in the event frame. An entity cannot seize or use a resource or material handler.

The separate experiment frame is very useful. It allows the modeler to separate the data and simulation commands from

the model structure. The user does not have to search through the model to find every occurrence of a variable. This makes it very easy to change variables and to see what variables there are. The modularity allows easy debugging and changes.

The macro-like capabilities ease typing and program length. This is very convenient since systems often have several almost identical elements.

SIMAN restricts which variables can be used for indexing. For example, to point to the parameter set only an integer value or A(I) is allowed. But, for the global variables, A(I) is not a valid index, but M is allowed. This can be very frustrating for the modeler, especially since different rules seem to apply for each type of indexing.

The material handling functions are easy to use and can simulate many material handling systems. This makes SIMAN especially useful for manufacturing simulations. I don't think any other general purpose simulation language has material handling functions.

SIMAN makes it very easy to get useful output. The user can specify what is to be output. This ranges from a full or partial trace, to a standard summary report, to a complete set of run data written to disk. In addition, specialized output can be generated in events. The output processor has many useful functions, including several ways to graphically show the data. It will convert from the internal data form to a form that can be read by most non-SIMAN programs. This

was designed for high resolution plotting, but the data can also be used by a statistical analysis program or other post-processor.

The interactive input of the model is a nice idea. It speeds things up by giving immediate feedback on errors, but it does not allow the translation step to be bypassed, which would speed things up even more. It is also completely compatible with batch input, so you can switch back and forth. This is especially useful for printouts, which are much easier to get in batch mode.

The output frame is also somewhat interactive. The frame could be aided greatly by menus and prompts for inputs. Sometimes there are a large number of parameters to enter for an output, but these must be entered as a long list with no feedback. If one mistake is made the entire command must be re-entered.

The thing that I feel is most lacking in SIMAN is that the experiment frame does not allow interactive input at run time. The ability to change experiment elements at run time would be very convenient and would save enormous amounts of time, especially on a personal computer, where the process of editing, translating, linking and reloading takes several minutes. If the run processor was interactive, changes would take just a few keystrokes.

The diagnostics generated are sometimes good and often quite bad. On the good side they offer what the proper syntax

is and give a list of possible fixes. On the bad side the error message is something like "Invalid Field." This means something is wrong someplace. No indication is given as to where the problem occurred in the line or how to fix it.

The documentation is in the form of a book, Introduction to SIMAN, and an installation guide. The documentation is usually fairly easy to follow. There are some problems in the description of the event frame. It is sometimes difficult to find what routines are available and how to use them.

A.8.2 Conclusion

Using SIMAN, it is usually easy to get the task done. A simple problem can be solved simply with a simple model. Very complex problems can still be solved using the same framework. The language is very flexible allowing several alternatives in modeling most problems. The modeler, not the language, decides the best way to model the system.

SIMAN has many annoying aspects to it. The most notable one is that the simulation is not interactive at run time. SIMAN seems headed in the right direction with the interactive model input and somewhat interactive output processor. Most of SIMAN's flaws can be gotten around with patience. The flexibility of the language allows many approaches to solving a problem and when one solution is not

allowed, it usually does not take long to find another solution.

The different frames of SIMAN combine to form a very powerful simulation language. Yet, it is still small and efficient enough to run on a moderately equipped personal computer with reasonable run times. (A PC with 256K RAM and two disk drives is needed). Also, the personal computer version is completely compatible with the mainframe version. This allows a very flexible working environment for SIMAN.

Overall, SIMAN provides a powerful and fairly easy to use environment for writing and running simulation models.

Appendix B

NOTES ON USING FORTRAN

This appendix lists a few observations on the use of FORTRAN on an IBM PC.

B.1 Incompatibility Between Different Versions of Microsoft FORTRAN

One thing that is very important to know is that the object files from different versions of Microsoft (MS) FORTRAN are not compatible. This is important when user written routines are to be linked to commercial precompiled object files or libraries. It is essential that the same version of FORTRAN be used to compile the user written routines as was used to compile the commercial software. The linker will usually print error messages when different versions of object code are used. It sometimes does not print any error messages, but the linked program will not run correctly.

The source code of the different versions of MS-FORTRAN are compatible for the most part. The defaults for the compilation are slightly different among the versions and the newer versions have some added features that were not in the older versions. So, if the source code is available for all of the routines to be compiled, there should not be much

difficulty in compiling and linking the program under one version of FORTRAN.

B.2 Compatibility Between Different Models of the IBM PC

The IBM PC (PC) and IBM PC/XT (XT) are completely compatible. Operations will be almost two times faster on the XT than on the PC because of the XT's hard disk, which is quicker than a diskette. Operations that do not use disks will run the same speed on both machines.

The IBM PC/AT (AT) is mostly compatible with the PC and XT when running MS-FORTRAN programs. As long as the AT has a math coprocessor, there does not seem to be any problem running programs compiled in MS-FORTRAN on the AT. If the AT does not have a math coprocessor, some programs will not run on the AT. If the programs were compiled in MS-FORTRAN 3.2, they can be relinked using the alternate math library and they will run correctly on the AT.

The AT is about four times faster than the PC and about two times faster than the XT.

B.3 Abnormal Exit from the Compiler or Linker

Often when a compile or link is aborted early, the scratch files on the disk are not closed properly and they take up disk space even though they do not show up in the

disk's directory. This often occurs when the compiler or linker stops because of insufficient memory or disk space. To recover the lost disk space the DOS CHKDSK program is used with the /F parameter specified. The file(s) created by CHKDSK, FILE00??.CHK, can be deleted from the root directory on the disk.

Appendix C

IBM PC USAGE

This appendix describes how the IBM PC was used to run the programs used in the research. The first section provides a description of the disk layout used and why. The rest of the sections describe how to run the programs.

C.1 Disk Layout

This section describes the disk layout used on PCs. The default drive is always used to store the user written programs and data. The other drive is used to store DOS and the application programs. A directory path is set up to point to the DOS and application programs' directories.

This disk layout reduces the number of times a drive needs to be specified to almost zero, almost eliminating the errors caused by specifying the wrong drive. The major disadvantage is that at least part of DOS should be stored on the user's disk. However, DOS does not need to be on any of the application disks.

This layout may be set up two ways. The first uses the initial default (boot) drive as the user drive. The other set up involves setting the default drive after the system is booted to the user drive. If the first method is used, the user disk must be bootable, taking up more disk space. This

method allows custom set up depending on which user disk is booted. The second set up forces the user to boot off the same disk all of the time, which can be an advantage because the initial set up will always be the same, eliminating the confusion caused by booting off a disk with an old AUTOEXEC.BAT.

I used the first set up during the research presented in this thesis. In retrospect, the second set up may be better, but until it has been used for quite a while it is hard to tell whether a set up is good.

I always used a directory structure on the user disks. This hides the DOS functions and startup files from the user directories. I never use a directory structure on the DOS and application disks because this causes problems when the disks are swapped, as they often are. DOS versions before 3.0 do not support a directory structure well enough to allow the application programs to be in a sub-directory. This sometimes causes problems when using a hard disk. A solution is to copy the programs that are needed the appropriate directory when needed, and erasing them after use. Another solution is to move either the application programs or user programs to diskette and running off the diskette as well as the hard disk. DOS 3.0 fixed this problem for the most part.

C.2 Translating and linking SIMAN models

I used three simple batch files to translate and link SIMAN models. They are identical to the steps suggested in the SIMAN Installation Manual for Microcomputers, except they run the SIMAN preprocessor listed in the appendix "SIMAN Preprocessor" on page 284 before the translation steps.

To translate the model issue the command:

```
MOD fn
```

with the disk labeled "SIMAN1" in drive B. The model statements must be in the file fn.MOD in the default directory. The output is placed in the file fn.M. The output of the preprocessor will be in the file fn.MO. The batch file is listed below.

```
B:SIMPRE %1.MOD %1.MO
B:MODEL %1.MO %1.M
```

To translate the experiment issue the command:

```
EXP fn
```

The file fn.EXP is read, preprocessed to fn.EX and output as fn.E. The batch file is:

```
B:SIMPRE %1.EXP %1.EX
B:EXPMT %1.EX %1.E
```

To link the two translated files together, use the command

```
LINKS fn
```

This invokes the following batch file.

```
B:LINKER %1.M %1.E %1.P
```

The output file is called fn.P and will be stored in the default directory.

C.3 How to run the High Level and Detailed Models

The high level model is contained in the files with file names of HL. They are stored in the directory /HL2. The first version of HL is in the directory /HL. These directories are on the disk labeled "FMS_HL". The detailed model is contained in the files with file names of FMS. They are stored in the directory /FMS on the disk "FMS". To run HL or FMS, the experiment must be edited to set up the desired parameters. If a plot output is desired then the unit numbers on the DSTAT element must be specified. If disk output is

created, no more than five runs can be made before the new output will start to overwrite output from previous runs. After the experiment has been set up as desired, the model and experiment are translated and linked as described above. The model does not need to be translated if it has not been changed since the last run.

The high level model is run by using the batch file HL. The batch file FMS is used to run the detailed model. Both of these batch files have no parameters. The batch files are

HL.BAT	FMS.BAT
B:FMS HL.P	B:FMS FMS.P

The default directory in the A drive must be \HL2 or \FMS, depending on which model is to be run. The disk with FMS.EXE must be in drive B. Currently the disk labeled "WORK" contains the FMS.EXE file.

When the program is started, a banner message is presented displaying the copyright notice. The program pauses to allow the user to read the message. Type a carriage return to continue with the run. The user is then prompted to see if any parameters need to be changed. To change any parameters, type in the parameter set number, the parameter number and the new parameter value (which may be real). The entries should be separated by blanks, not commas. Enter all three parameters before pressing the carriage return. As many

parameters may be changed as desired. The changes remain in effect in any following runs in this invocation of SIMAN. Only values in the parameter set may be changed. A new (undefined) parameter may not be specified. To start the run enter a zero followed by a carriage return.

Any trace and summary report output will then be sent to the screen. To send the output to the printer type a Ctrl-PrtSc before the run starts. To send the output to a file, invoke the simulation with the command

```
B:FMS modn.P > fn.ext
```

where modn is either HL or FMS, and fn.ext is the file the trace and summary report are sent to. When the output is redirected in this manor, nothing shows up on the screen, so all of the inputs must be typed blindly.

C.4 Running the Plot Program

Before the plot program can be run, the data created during the run must be converted to a form that can be read by the plot program. This is done using the OUTPT program that comes with SIMAN. After the data is converted the plot program is run. Use the batch program CP to convert the data from the first run and start the plot program. The batch file is

```
B:OUTPT.EXE <DATA.CRE
PAUSE Insert the WORK disk in drive B
B:PLOTS
```

The "SIMAN1" disk, which contains the OUTPT program, must be in the B drive when this batch file is used.

The DATA.CRE file contains the OUTPT commands to convert the data from the first run to the data that can be read by the plot program. The DATA.CRE file is

```
BEGIN;
EZPREP: 20, 76, 77:
        25, 78, 79:
        30, 80, 81:
        35, 82, 83:
        40, 84, 85;
EZPREP: 45, 86, 87:
        50, 88, 89:
        55, 90, 91:
        60, 92, 93:
        65, 94, 95:
        70, 96, 97;
END;
```

To plot the data from runs other than the first run, the first column of DATA.CRE must be changed to point to the data from the run to be plotted. To plot the second run the first column is changed to 21, 26, 31, ..., 71. To plot the fifth run the column is changed to 24, 29, 34, ..., 74. The last two columns should not be changed because PLOT expects the data to be in the files OUTPUT.76 through OUTPUT.97.

A sample session of the plot program is shown in Figure C.1 on page 165. The loading of the data takes

several minutes. Each time a file is loaded a "." is printed. The help menu is then automatically printed. The current parameters are shown on the line that starts "Current:". The first position indicates the type of plot (M for medium resolution screen plot, H for high resolution monochrome screen plot, P for plot output on an IBM 749 pen plotter and Q for quit the program without plotting). The second and third position show the beginning and ending time of the plot. The "T=" position shows the default amount of time to be plotted. The last position indicates whether or not a y-axis label will be plotted (Y for y-axis label plotting and N for no y-axis label plotting. After the status line is printed, the user is prompted for an input. To make the plot as indicated by the status, type a carriage return. If any of the parameters are to be changed, type in the command to change the parameter. Once all of the parameters are as desired, type a carriage return to see the plot. After the plot is done, the default plot time is set to start at the time the last plot ended and be for T minutes.

A plot can be printed on the IBM Graphics Printer. The DOS GRAPHICS command must be issued before the plot program is started. This command only needs to be issued once per boot. To plot to the printer, display a monochrome high resolution plot on the screen and press shift-PrtSc. The plot on the screen will be copied to the printer.

```
Inputting data.....................
Valid inputs are:
  <CR>  to draw a plot
  # [#]  to set the begin [and end] plot time[s]
  M  for medium resolution color screen plots
  H  for high resolution B&W screen plots
  P  for plotter plots
  Q  to quit the program
  Y  for y-axis labels
  N  for no y-axis labels
  T#  to set the time per plot
  B#  to set the beginning plot time
  E#  to set the ending plot time
Current: M B=    .0 E=  30.0 T=  30.0 Y
Enter Option (X for help):  b
Enter the begin plot time: 10

Current: M B=  10.0 E=  40.0 T=  30.0 Y
Enter Option (X for help):  T
Enter the time per plot:   60

Current: M B=  10.0 E=  70.0 T=  60.0 Y
Enter Option (X for help):  e30.5

Current: M B=  10.0 E=  30.5 T=  60.0 Y
Enter Option (X for help):  p

Current: P B=  10.0 E=  30.5 T=  60.0 Y
Enter Option (X for help):  <cr>

     (Pen plotter plot is drawn from time 10.0 to 30.5)

Current: P B=  30.5 E=  90.5 T=  60.0 N
Enter Option (X for help):  q

Current: Q B=  30.5 E=  90.5 T=  60.0 N
Enter Option (X for help):  <cr>

Current: Q B=  30.5 E=  90.5 T=  60.0 N
Stop - Program terminated.
```

Figure C.1 Sample PLOT Session

There are sometimes problems with initialing the pen plotter from this program. The plotter can be initialized by running a sample plot from another software package before starting the PLOT program.

C.5 Compiling FORTRAN

FORTRAN programs are compiled as described in the FORTRAN users manual. I never generated a listing file because it adds time to the compilation, takes up disk space and provides very little additional information. The error messages are all printed to the screen with the line number where the error is. This is usually quite sufficient for removing the compile errors. A useful batch file for running both passes of the compiler is listed below.

```
B:FOR1 %1;
IF EXIST PASIBF.BIN B:FOR2
```

The version of SIMAN used requires that any routines that are to be linked to SIMAN be compiled in Microsoft Version 3.1 FORTRAN. Any other version of FORTRAN will not work!

The plotting program must be compiled in the same version of FORTRAN that the Grafmatic and Plotmatic libraries were compiled in. The FORTRAN version used in this research

was TI FORTRAN (Microsoft 3.0). It should be very simple to upgrade the plot program to Microsoft 3.2 FORTRAN.

C.6 Linking FORTRAN Routines to SIMAN

To use any FORTRAN routines in the model, the routines must be linked to the SIMAN object code to form a new SIMAN run file. This section describes how this is done.

The routines to be linked must first be compiled using the same version of FORTRAN as was used to compile the SIMAN routines. The "SIMAN Installation Manual" will say what version of FORTRAN was used to compile the SIMAN routines. The version of SIMAN used in this research was compiled with Microsoft 3.1 FORTRAN. The batch file LFMS is then used to link the SIMAN routines to the modeler written routines to form FMS.EXE which is used to run both the high level and detailed models. The batch file is

```
B:LINK EVENT+TRACE+PRIME+@B:SIMAN.RSP,B:FMS/PAUSE,NUL,B:
```

The modeler written routines must be in the default directory in the A drive. These routines are in the directory \FMSFOR on the disk "SIMFOR". The "SIMAN_LIB" disk must be in the B drive. This disk contains the 3.1 FORTRAN library and all of the SIMAN object modules. The SIMAN.RSP file is a slightly modified version of the SIMAN.RSP file that comes with SIMAN.

It is modified to look for the SIMAN object modules on the B drive. The SIMAN.RSP file used is listed below.

```
B:SIMAN+B:SEXEC+B:SEXECA+B:SBLOCK+B:SUTIL1+B:SUTIL1A+
B:SUTIL2+B:SUTIL2A+B:SUTIL3+B:SUTIL3A+B:SUTIL3B+B:RA+
B:FNAME+B:XBAN+B:BAN+B:DLIB+B:RANDOM
```

Make sure there is at least 20 or 30k bytes free on the default (A) drive before attempting the link. The default drive is used for temporary storage during the link. The link takes about six minutes. There will be several error messages about modules being multiply defined, ignore these errors. At the end of the link the linker will prompt the user to insert the disk the .EXE file will be written onto. At this point put the "WORK" disk in the B drive and type a carriage return. After the FMS.EXE file is written, it will be usable to run the models.

C.7 Linking the Plot Program

To link the plot program the LPLOT batch file is used after each of the source code files have been compiled. The LPLOT batch file is

```
B:LINK PLOT+INPUT+SCREEN+ROBOT+MACH+PLOTTER,
     B:/PAUSE,NUL,B:GRAFTI+B:PLOTTI+B:+
```

This file needs to be on one line; it is split here so it will fit within the margins. The object files must be in the default directory on the A drive. They are in the directory \HLPLOT on the disk "SIMFOR". A disk with the FORTRAN link program must be in the B drive. The "TI_FORTRAN" disk is good for this. The last plus in the LPLOT batch file causes the link program to stop and prompt for more libraries. At this point change the disk in the B drive so the "FOR_LIBS" disk is in the B drive. This disk has the FORTRAN, GRAFMATIC and PLOTMATIC libraries on it. Since the extra plus in the batch file was to cause a pause so the disk could be changed, no additional library needs to be specified. Just type a carriage return to start the linker.

The link takes a long time, so sit back and relax. When the link is done, the linker will prompt for the disk the .EXE file is to be written onto. At this point put the "WORK" disk in drive B and type a carriage return. When the PLOT.EXE file has been written to disk, it is ready to be used.

The current version of the plot program does not have the routines for using the pen plotter. This reduces the size of the .EXE file and speeds up loading. To link the plot routines for this setup the batch file LS is used. The LS batch file is

```
B:LINK PLOT+INPUT+SCREEN+ROBOT+MACH,B:PLOTS/PAUSE,
     NUL,B:GRAFTI+B:+
```

Again, this file must fit on one line. The link is performed in the same way as for the full plot program. Ignore the errors about not being able to find any object modules, we have left them out deliberately! This version of the plot program is called PLOTS. It runs the same way as the other plot program, except it will not drive a pen plotter.

There is also a batch program, LP, that will create a plot program that will only drive the pen plotter.

C.8 Conclusion

This appendix gives an overview of how to run the programs developed during this research. Many more batch files could be created for specific needs. A possible batch file would be one that edits the experiment, translates the experiment, links the experiment and model, runs the model, converts the output, starts the plot program and then starts all over. This would be useful if a large number of runs are to be made in a row. The user would only need to swap disks when needed and type in the changes to the experiment between runs. One thing that must be kept in mind here is that DOS does not allow the nesting of batch files.

All of the batch files listed in this appendix will only work with diskettes. When using a hard disk the batch files must be rewritten so that instead of pointing to a drive the batch files point to a directory. This is much easier to do

when using DOS 3.0 or higher. When using DOS 2 the directory structure will have to be a little different so all of the related object files and libraries will be in the same directory. A path must be set up to any program used because the directory can not be specified at run time. The other major difference between the diskette and hard disk set up is that some of the larger directories can be combined on the hard disk. The "SIMAN1" and "WORK" disks are combined to make a \SIMAN directory. All of the libraries and the SIMAN object files can be placed in the same directory as the FORTRAN compiler that they are associated with.

Appendix D

HIGH LEVEL MODEL LISTING

D.1 HL.DES

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$                                                                              $
$                                                                              $
$                                   HL.DES                                     $
$                                                                              $
$                                     by                                       $
$                                                                              $
$                                 Tim Martin                                   $
$                                                                              $
$                     11/30/84                    Ver 2.25                     $
$                                                                              $
$                                                                              $
$       This is a description of a model to simulate the Virginia Tech         $
$       Flexible manufacturing system at a high level.  The model is in        $
$       the files HL.MOD and HL.EXP, and the FORTRAN subroutines are in        $
$       the files PRIME.FOR and EVENT.FOR.                                     $
$                                                                              $
$                                                                              $
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

This model simulates the Virginia Tech Miniature Flexible Manufacturing System (FMS) at a high level. The FMS consists of four machining stations, an input conveyor and an exit conveyor. All parts are mounted on a pallet before they enter the system. Pallets are moved around the system by a robot in the center of the system. The machines and conveyors are equally spaced around the robot. The layout is shown below. The robot has a capacity of one pallet. All

```
                  Machine D        Exit Conveyor
                          \        /
                           \      /
             Machine C ------ Robot ------ Input Conveyor
                           /      \
                          /        \
                  Machine B        Machine A
```

machines are different and can not be substituted for any other machine. A pallet can only move to a machine that is in increasing alphabetical order from the machine the pallet is currently at. This is to prevent a deadlock from occurring. If, for example, the pallet at machine A needs to go to machine B and the pallet at machine B needs to go to machine A a lock up will occur and the

system will stall. The machine sequence for any pallet type is fixed. The processing time for a pallet type on a particular machine is fixed. After processing is done on a machine, the machine is brought to its home position. This time may be considered as an extra random time or it may be considered as part of the fixed processing time. (The time should be almost the same from pallet to pallet.) There may be any number of pallet types. Specifics about the model are explained below.

The model is written in the simulation language SIMAN. The model consists of three parts: the SIMAN model frame, the SIMAN experimental frame and FORTRAN subprograms. The model frame, stored in HL.MOD, contains the model. The experimental frame, stored in HL.EXP, contains the 'experiment' - or data. The FORTRAN subprograms, stored in PRIME.FOR and EVENT.FOR, are used to input parameters at the beginning of a run and to generate a specialized trace of the runs. The model has SIMAN statements that describe how the FMS is connected together and how a pallet is to move through the system. The experiment has SIMAN statements that control the simulation and has statements that define the variables used in the model. The control statements tell SIMAN how long the simulation is to last, the maximum values for many variables, whether or not to trace part of the simulation and which variables should be used for statistical output. The FORTRAN subprograms are mostly just I/O statements. The FORTRAN subprograms must be linked to the SIMAN processor before they can be used.

The model consists of 6 stations:

Station 1 is the input conveyor,
Stations 2-5 are the machine stations, and
Station 6 is the exit conveyor.

To move a pallet from one station to another, the robot is requested at the current station. When the robot arrives, a pick is performed. A pick tells the robot to pick up the pallet. The pick is modeled simply as a time delay. The robot is then moved to the desired station. After arriving at the station, a place is performed. A place tells the robot to place the pallet on the machine or conveyor. The place is modeled by a time delay. The robot is then released so other pallets can use the robot.

The attributes for all pallets are defined as:

A(1) is the time after the start of the simulation that the pallet entered the system.
A(2) is the type of pallet plus 10. The pallet type defines the sequence of machines and processing time for the pallet.
A(3) is the station the pallet is at.
A(4) is the processing time at station A(3) - This may or may not include the time it takes for the machine to go to its home position after processing is done. The time to get to its home position can be represented as a uniform distribution.

A(5) is the number of work stations the pallet has been to. The input and exit conveyors count for this value.
A(6) is used as an index when getting the next processing time and in finding the time it takes a machine to get to its home position.
A(7) is the pallet number - the pallets are numbered in the order they entered the system, starting at one.
A(8) is the station that a pallet is moving to.

The resources MACHINE(1-6) are used to control the availability of all of the stations. If the resource is unavailable, the machine is busy with a part or has a part coming to it. If the resource is available, then the pallet can seize the resource and request the robot to move it to the station. The output conveyor is always available when the robot is free.

The variable X(1) is used to keep track of the number of pallets on the input conveyor. The number of pallets on the input conveyor is equal to the sum of the number of pallets in QUEUE(1) and the number of busy units of MACHINE(1). MACHINE(1) is the input conveyor, and QUEUE(1) is the queue for pallets waiting for the input conveyor.

The variables X(2-5) are used to indicate that stations 2-5 (machines A-D) are busy. If X(i) is 1, the station is busy with a part. These variables are used only to keep statistics on machine utilization. The difference between X(i) utilization and MACHINE(i) utilization is the amount machine i is waiting for a pallet to arrive or waiting for a pallet to be removed.

The variable X(7) is used to hold the station the robot is at, or the last station the robot was at. This variable is used only for data collection.

Signal code 1 is used to signal when the robot is finished with a move.

The queues used in the model are:

1-6 Wait for station q to become available
7 Wait for the robot to pick up a pallet
8 Wait for both the station and robot to become available. This queue represents jobs on the task list that need to be processed.

The parameter sets are defined as:

1 is the pallet arrival time mean (1,1).
2 is the cumulative probability that a pallet is of a certain type. The odd values in this set are the cumulative probabilities of the pallet types in the even values. There must be an entry in this set for all possible types of pallets.
3 contains the trace status. If (3,1) is 1, the trace is on. If (3,1) is 0, the trace is off.

The rest of parameter set 3 is a sequence of times that represent the times when the trace status changes. There must be a change at or after the end of the run.

4 is the pick and place times. (4,1) is the pick time and (4,2) is the place time.

5-8 are the minimum and maximum time it takes to get a machine to its home position. Parameter set 5 is used for machine A, 6 for B, 7 for C and 8 for D. (i,1) is the minimum, and (i,2) is the maximum. If both of these values are set to zero, then the time to bring the machine to its home position should be included in the processing time for each machine.

9-10 are currently unused.

11 up are used to define the machine sequences and processing times on each machine for each pallet type. The set used for each pallet type is the pallet type number plus 10.
(i,1-5) are used for the part sequence. All parts start at station 1, so this is assumed and must not be included. Each part must end at station 6, and this must be included in the part sequence.
(i,6-10) are used for the processing time on the station stored in j-5. A processing time for the exit conveyor must be specified (it does not matter what it is). The processing time may or may not include the time it takes for the machine to go to its home position after processing is done. See parameter set 3 for details.

The random number streams used are:

1 is used for the time between pallet arrivals,
2 is used for the time it takes a machine to get to its home position,
3 is used to determine the part type.

Counter number 1 is used to count the number of pallets that have entered the system. This is used to mark the pallet number (A(7)).

The trace is generated by calls from the model frame to the FORTRAN subprograms. The calls are made using the SIMAN command EVENT: i, where i corresponds to the appropriate subprogram. There is a subprogram called EVENT that maps the SIMAN event numbers to FORTRAN subprograms. The trace is turned on and off using an additional FORTRAN event (Event number 6). An entity is created in PRIME that is scheduled to arrive at event 6 the next time a trace status change is to occur. Whenever the entity arrives at the event it changes the trace status and gets a new trace change time from parameter set 3. The entity's attribute 1 is used to point to the next time delay in parameter set 3.

Also in PRIME, is a subprogram to input any changes the user wants in the parameter set. Only values in the parameter set can be changed. The change is valid only for the current run.

D.2 HL.MOD

```
BEGIN;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                              ;
;                                                                              ;
;                                   HL.MOD                                     ;
;                                                                              ;
;                                     by                                       ;
;                                                                              ;
;                                 Tim Martin                                   ;
;                                                                              ;
;                    11/30/84                    Ver 2.18                      ;
;                                                                              ;
;                                                                              ;
;           Model to simulate the Virginia Tech FMS at a high level.           ;
;                          See HL.DES for details.                             ;
;                                                                              ;
;                                                                              ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                              ;
;                                                                              ;
;            Pallet arrival block                                              ;
;                                                                              ;
;                                                                              ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
        CREATE: EX (1,1):-        Pallets arrive based on an exponential
-                                 distribution (The mean is stored in P(1,1) )
        MARK( 1 );                Store the arrival time in A(1)
        COUNT: 1;                 Count the number of pallets that have arrived
        ASSIGN: A(7) = NC(1);     Mark this as the nth pallet
        ASSIGN: M = 1;            The input conveyor is station 1
        ASSIGN: A(3) = 1;         The pallet is at the input conveyor
        ASSIGN: A(2) = DP(2,3) + 50;  Find out which pallet type it is. The
;                                 types are defined by the discrete
;                                 distribution table found in parameter set 2
        ASSIGN: X(1) = NQ(1) + NR(1) + 1;  Compute the number of pallets on the
;                                 input conveyor (include the arriving pallet)
        EVENT: 1;                 Trace event 1 - pallet arrival
;
        QUEUE, 1;                 Wait on the input conveyor until the
        SEIZE: MACHINE(1);            end of the conveyor is reached
;
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                              ;
```

```
;                                                                          ;
;           The following block is used to move to the next station        ;
;                                                                          ;
;                                                                          ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
MOVEON   ASSIGN: A(5) = A(5) + 1;          Increment the sequence number
         ASSIGN: A(8) = P( A(2), A(5) );   Get the next station number
         ASSIGN: A(6) = A(5) + 5;          Index the processing time
         ASSIGN: A(4) = P( A(2), A(6) );   Get the next processing time
         ASSIGN: M = A(8);         The next machine must be a function of M
         EVENT: 2;                 Trace event 2, a pallet is ready to move to
;                                  another station
;
;
CHKSTARB BRANCH, 1:-               Check to see if both the next station and
-                                  robot are available
              IF, ( NR(M) .LT. MR(M) ) .AND. ( NT(1) .EQ. 0 ),
                   GETSTARB:-      If so, go to 'GET STation And RoBot'
              ELSE, WT4STARB;      If not, go to 'WaiT FOR STation And RoBot'
;
;
WT4STARB QUEUE, 8;                 Wait for both the station and robot to
;                                  become available
         WAIT: 1:-                 The station can only become available after
-                                  the robot has finished a move
         NEXT( CHKSTARB );         Every time the robot finishes a move, check
;                                  to see if any of the pallets can move on
;
GETSTARB QUEUE, M;                 Wait for the next machine to
         SEIZE: MACHINE( M );           become available
         ASSIGN: M = A(3);         Restore the current station number
;
;
         QUEUE, 7;                 Wait for the robot
         REQUEST: ROBOT;
         ASSIGN: X(7) = M;         The robot is at station M
         EVENT: 3;                 Trace event 3 - a pallet is picked up by
;                                  the robot
;
         DELAY: P(4,1);            Perform a pick
         ASSIGN: X(7) = M+.5;      The robot is leaving station M
;
         RELEASE: MACHINE(M);      The old machine is now available
         ASSIGN: X(1) = NQ(1) + NR(1); Compute the number of pallets on the
;                                       input conveyor
;
         TRANSPORT: ROBOT, A(8);  Move the pallet to its next station
;
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;                                                                          ;
;                                                                          ;
;          The machine stations                                            ;
;                                                                          ;
;                                                                          ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
        STATION, 2-5;
        ASSIGN: X(7) = M;         The robot is at the current station
        ASSIGN: A(3) = M;         Set A(3) to the current station
;
        DELAY: P(4,2);            Perform a place
        ASSIGN: X(7) = M+.25;     The robot is done at station M
;
        FREE: ROBOT;              Done with the robot for now
        EVENT: 4;                 Trace event 4 - arrival at a machine station
        SIGNAL: 1;                Tell the world that the robot is done
;
        ASSIGN: X(M) = 1;         Set the machine to busy.  Used to find
;                                      the machine utilization.
        DELAY: A(4);              Processing time
;
        ASSIGN: A(6) = M + 3;     Set A(6) to the parameter set that has
;                                      the data on how long it takes to
;                                      bring the machine to home
        DELAY: UN( A(6), 2);      Delay by the time it takes to bring the
;                                      machine to its home position
        ASSIGN: X(M) = 0:-        Done with the machine
-
        NEXT( MOVEON );           Move on to the next station
;
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                          ;
;                                                                          ;
;          Station 6 is the exit conveyor                                  ;
;                                                                          ;
;                                                                          ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
        STATION, 6;
        ASSIGN: X(7) = 6;         The robot is at the exit conveyor
;
        RELEASE: MACHINE( 6 );    The output conveyor is not committed
;
        DELAY: P(4,2);            Perform a place
        ASSIGN: X(7) = 6.25;      The robot is done at the exit conveyor
;
        FREE: ROBOT;              Done with the robot
        EVENT: 5;                 Trace event 5 - A pallet is leaving the system
```

```
        SIGNAL: 1;              Tell the world that the robot is done
;
        TALLY:1, INT(1):-       Get time in the system
        DISPOSE;                The present pallet is done
;
;
END;
```

D.3 HL.EXP

```
BEGIN;
PROJECT, FMS HL MV2.18 EV2.20, Tim Martin,  5/ 7/85;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                      ;
;                                                                      ;
;                              HL.EXP                                  ;
;                                                                      ;
;                                by                                    ;
;                                                                      ;
;                            Tim Martin                                ;
;                                                                      ;
;                 5/ 7/85                  Ver 2.20                    ;
;                                                                      ;
;                                                                      ;
;         This experiment goes with the model that simulates the       ;
;       Virginia Tech FMS at a high level.  See HL.DES for details.    ;
;                                                                      ;
;                                                                      ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;
DISCRETE, 50, -The maximum number of concurrent entities
           8, -The maximum number of attributes per entity
           8, -The number of files
           6; -The number of stations
;
REPLICATE, 1, -The number of runs
    0,  7200; -The run time - Base is 1 second
;
;
RESOURCES: 1-6, MACHINE,  1,1;-Define the machines
;
TRANSPORTERS: 1, ROBOT,        -Transporter 1 is a robot
         1,                    -with capacity 1
         1,                    -using distance set 1
         0.608,                -velocity
         1-A;                  -starting at station 1, ready for service -
                               -(Active)
;
COUNTERS: 1, PALLET NUMBER;    -Counter number one keeps track of the number
                               -    of pallets that have entered the system
;
;
DISTANCES: 1, 1-6,             -Distance set 1 for all stations (1-6)
                               -     Gives the distance between all stations
    10,20,30,20,10/            -Station 1 to station 2, 3, 4, 5 and 6
       10,20,30,20/            -Station 2 to station    3, 4, 5 and 6
          10,20,30/            -Station 3 to station       4, 5 and 6
             10,20/            -Station 4 to station          5 and 6
```

```
               10;          ~Station 5 to station               6
;
PARAMETERS:   1, 250:       ~Pallet arrival time mean
~
~       Cumulative probability that a pallet is of a certain type
            2, .25, 1,      ~25% are type 1
               .50, 2,      ~25% are type 2
               .75, 3,      ~25% are type 3
              1.00, 4:      ~25% are type 4
~
            3,   0,         ~The initial trace status ( 0 = No Trace,
                            ~    1 = Trace On )
             15000,18000,   ~The time when the trace status changes.
             18000,18000,   ~    There must be a change listed at or
             18000,18000,   ~    after the end of the run. There may
             18000,18000:   ~    be any number of changes during a run
~
            4,  23.25,      ~The pick time
                22.5:       ~The place time
~
            5,   0, 0:      ~The minimum and maximum time it takes to
                            ~    get to machine A's home position
~
            6,   0, 0:      ~The minimum and maximum time it takes to
                            ~    get to machine B's home position
~
            7,   0, 0:      ~The minimum and maximum time it takes to
                            ~    get to machine C's home position
~
            8,   0, 0:      ~The minimum and maximum time it takes to
                            ~    get to machine D's home position
~
~       Parameter sets 9-50 are unused
~
~       Define the machine sequences and processing times on each machine
~       for the different pallet types
           51,  2,  3,  4,  6,  0,   ~Machine sequence for pallet type 1
               190, 120, 240, 0:     ~Processing times for pallet type 1
~                190 seconds on machine 2, 120s on machine 3, 240s on
~                machine 4 and then exit to machine 6 -the exit conveyor
           52,  2,  6,  0,  0,  0,   ~Machine sequence for pallet type 2
               190, 0:               ~Processing times for pallet type 2
           53,  3,  6,  0,  0,  0,   ~Machine sequence for pallet type 3
               120, 0:               ~Processing times for pallet type 3
           54,  4,  6,  0,  0,  0,   ~Machine sequence for pallet type 4
              240,  0;               ~Processing times for pallet type 4
~       More pallet types can be added by changing the cumulative
~       probabilities of the different pallet types to include the new
~       pallet type(s), and then adding the machine sequence/processing
~       times to the end of the parameter list.
;
```

```
;
RANKINGS: 1-8, LVF(1);  ~Process the pallets based in the order in which
~                        they came into the system.  (Order the entities
~                        in the queues based on low value of attribute 1
~                        first.)
;
;
TALLIES: 1, IN SYSTEM TIME;  ~For each finished pallet, find out how long
~                             it was in the system
;
;        Collect statistics on the following time varying parameters:
;             Output the changes to the file numbers specified - OUTPUT.nn
DSTAT:   1,  X(2), Station A Util., 20:
         2, NR(2), Station A Commit, 25:
         3,  X(3), Station B Util., 30:
         4, NR(3), Station B Commit, 35:
         5,  X(4), Station C Util., 40:
         6, NR(4), Station C Commit, 45:
         7,  X(5), Station D Util., 50:
         8, NR(5), Station D Commit, 55:
         9, NT(1), Robot Utilizatn, 60:
         10, X(1), # on Conveyor, 65:
         11,NQ(8), #Wait4 Rbt & Stn:
         12, X(7), Robot Position, 70; ~Used only for plotting
;
;
INITIALIZE, X(7)=1.2;   ~The robot starts at station number 1
;
;
SEEDS:        ~Initialize the random number streams:
    1,374,:        ~Stream 1 is used for the time between pallet arrivals
    2, ,:          ~Stream 2 is used for the time it takes a machine to
                   ~    get to its home position
    3,766,;        ~Stream 3 is used to determine the part type
;
;
END;
```

Appendix E

HIGH LEVEL PLOT LISTINGS

E.1 PLOT.FOR

```
$STORAGE:2
C
C
C   PLOT.FOR
C
C   by
C
C   Tim Martin
C
C   11/ 8/84
C
C   This plots the output data from the SIMAN model HL2
C
C



C
C
C       The main program calls the subroutines necessary to do the
C       plotting
C
C       Ver  1.05   10/31/84
C
C

$INCLUDE: 'PLOTCOM.FOR'


C       Initialize the plotting options
      TYPE = 5
      BPTIME = 0.0
      TMPSCR = 30.0
      EPTIME = 0.0
      YLABEL = 'Y'

C       Input and convert the raw data into plot data
      CALL INPUT

C       Prompt the user to see if he wants to continue, and if so,
C       what he wants done next
 10   CALL PROMPT
```

```
C       Do what the user wants
      GOTO ( 20, 20, 30, 40 ) TYPE

C       Put a plot on the screen
 20   CALL SCREEN
      GOTO 10

C       Put a plot on the IBM 749 plotter
 30   CALL PLOTER
      GOTO 10

C       Clear the screen and quit
 40   CALL QSMODE(2)
      STOP

      END



      SUBROUTINE PROMPT
C
C
C       PROMPT allows the user to say what he wants to do next.
C       The options are a medium resolution screen plot, a high
C       resolution screen plot, a plotter plot and to quit.  If
C       the user is not quitting he can enter the time period he
C       wants to plot.  Or he can plot the next default time period
C       or he can change the defaults.
C
C       Ver  2.03   11/ 2/84
C
C

$INCLUDE: 'PLOTCOM.FOR'

      REAL BPMIN, EPMIN

      LOGICAL*2 CHANGE, SETBPT, SETEPT

      CHARACTER*80 LINE


C       Initialize the data for the input
      BPMIN = BPTIME/60.
      EPMIN = EPTIME/60.
      SETBPT = .FALSE.
      SETEPT = .FALSE.

C       Check to see what type the last plot was
      IF ( TYPE .NE. 3 ) THEN
```

```
C               It was a screen plot
            IF ( TYPE .EQ. 5 ) THEN

C                   This is the first plot, print the options allowed
                TYPE = 1
                WRITE( LINE, '(A80)' ) 'X'
                CALL GETIN( LINE, CHANGE, SETBPT, SETEPT, BPMIN, EPMIN )

            ELSE

C                   Move the cursor to the bottom of the screen
                CALL QCMOV( 0, 1 )

C                   Input on the plot screen without a prompt
                READ(*, '( A80 )' ) LINE
                CALL GETIN( LINE, CHANGE, SETBPT, SETEPT, BPMIN, EPMIN )

            ENDIF

        ELSE

C               The last plot was a pen plotter plot, print the status
            WRITE( LINE, '(A80)' ) 'P'
            CALL GETIN( LINE, CHANGE, SETBPT, SETEPT, BPMIN, EPMIN )

        ENDIF

C           Input options until just a carriage return is entered
 10     WRITE(*, 100 )
 100    FORMAT( ' ', 'Enter Option (X for help):  ', \ )
        READ(*, '(A80)' ) LINE
        CALL GETIN( LINE, CHANGE, SETBPT, SETEPT, BPMIN, EPMIN )

C           Check to see if an option was entered
        IF ( CHANGE ) GOTO 10

C           Check to see which defaults to use
        IF ( ( .NOT. SETBPT ) .AND. ( .NOT. SETEPT ) ) BPMIN = EPMIN
        IF ( ( .NOT. SETBPT ) .AND. ( SETEPT ) ) BPMIN = EPMIN - TMPSCR
        IF ( .NOT. SETEPT ) EPMIN = BPMIN + TMPSCR

        BPTIME = BPMIN*60
        EPTIME = EPMIN*60

        RETURN

        END
```

```
      SUBROUTINE GETIN( LINE, CHANGE, SETBPT, SETEPT, BPMIN, EPMIN )
C
C
C     GETIN interprets LINE to see if any valid commands were entered.
C     If a valid command was entered the command is processed and
C     CHANGE is returned TRUE.  If an invalid command is entered, a
C     list of valid commands is printed and CHANGE is returned TRUE.
C     If just a blank line is entered CHANGE is returned FALSE.
C
C     Ver:   1.01   10/31/84
C
C

$INCLUDE: 'PLOTCOM.FOR'

      REAL BPMIN, EPMIN, OBPMIN, OEPMIN

      INTEGER*2 PTR, EPTR, PTR1

      CHARACTER*80 LINE

      CHARACTER*1 CHR(80), CTYPE

      LOGICAL*2 CHANGE, SETBPT, SETEPT


      CHANGE = .TRUE.

C          Convert the line input into an array of characters
      READ( LINE, '(80A1)' ) ( CHR(PTR1), PTR1 = 1, 80 )

C          Find the first non-blank character
      DO 10 PTR = 1, 80
 10       IF ( CHR(PTR) .NE. ' ' ) GOTO 20

C          The line was blank, return with CHANGE equal FALSE
      CHANGE = .FALSE.
      GOTO 30

C          Decode the option
 20   IF ( ( CHR(PTR) .EQ. 'Y' ) .OR. ( CHR(PTR) .EQ. 'y' ) ) THEN

C               Put y-axis labels on the plot
           YLABEL = 'Y'

      ELSEIF ( ( CHR(PTR) .EQ. 'N' ) .OR. ( CHR(PTR) .EQ. 'n' ) ) THEN

C               Don't put y-axis labels on the plot
           YLABEL = 'N'

      ELSEIF ( ( CHR(PTR) .EQ. 'M' ) .OR. ( CHR(PTR) .EQ. 'm' ) ) THEN
```

```
C               Go to  medium resolution color screen plot mode
            TYPE = 1

      ELSEIF ( ( CHR(PTR) .EQ. 'H' ) .OR. ( CHR(PTR) .EQ. 'h' ) ) THEN

C               Go to high resolution black and white screen plot mode
            TYPE = 2

      ELSEIF ( ( CHR(PTR) .EQ. 'P' ) .OR. ( CHR(PTR) .EQ. 'p' ) ) THEN

C               Go to pen plotter plot mode
            TYPE = 3

      ELSEIF ( ( CHR(PTR) .EQ. 'Q' ) .OR. ( CHR(PTR) .EQ. 'q' ) ) THEN

C               Quit the run
            TYPE = 4

      ELSEIF ( ( CHR(PTR) .EQ. 'T' ) .OR. ( CHR(PTR) .EQ. 't' ) ) THEN

C               Change the default time per plot
            CALL GETNUM( CHR, PTR+1, 'time per plot:   ', TMPSCR, EPTR )

      ELSEIF ( ( CHR(PTR) .EQ. 'B' ) .OR. ( CHR(PTR) .EQ. 'b' ) ) THEN

C               Set the begin plot time
            CALL GETNUM( CHR, PTR+1, 'begin plot time: ', BPMIN, EPTR )
            SETBPT = .TRUE.

      ELSEIF ( ( CHR(PTR) .EQ. 'E' ) .OR. ( CHR(PTR) .EQ. 'e' ) ) THEN

C               Set the end plot time
            CALL GETNUM( CHR, PTR+1, 'end plot time:   ', EPMIN, EPTR )
            SETEPT = .TRUE.

      ELSEIF ( ( CHR(PTR) .GE. '0' ) .AND. ( CHR(PTR) .LE. '9' ) ) THEN

C               Set the begin and maybe the end plot time(s)
            CALL GETNUM( CHR, PTR, 'begin plot time: ', BPMIN, EPTR )
            SETBPT = .TRUE.
            CALL GETNUM( CHR, EPTR, 'RETURN           ', EPMIN, PTR )
            IF ( PTR .LE. 80 ) SETEPT = .TRUE.

      ELSE

C               Print the options available
            WRITE(*,100)
 100        FORMAT( '0', 'Valid inputs are:' / ' ', '  <CR>  to draw a ',
     1          'plot' / ' ', '  # [#]  to set the begin [and end] ',
     2          'plot time[s]' / ' ', '  M  for medium resolution color'
```

```
     3          ,' screen plots' / ' ', '  H  for high resolution B&W ',
     4          'screen plots' / ' ', '  P  for plotter plots' / ' ',
     5          '  Q  to quit the program' / ' ', '  Y  for y-axis ',
     6          'labels' / ' ', '  N  for no y-axis labels' / ' ',
     7          '  T$  to set the time per plot' / ' ', '  B$  to set ',
     8          'the beginning plot time' / ' ', '  E$  to set the ',
     9          'ending plot time' )

      ENDIF

C          Print the current status
C          Check to see which defaults to use
 30   OBPMIN = BPMIN
      OEPMIN = EPMIN
      IF ( ( .NOT. SETBPT ) .AND. ( .NOT. SETEPT ) ) OBPMIN = OEPMIN
      IF ( ( .NOT. SETBPT ) .AND. ( SETEPT ) ) OBPMIN = OEPMIN - TMPSCR
      IF ( .NOT. SETEPT ) OEPMIN = OBPMIN + TMPSCR

C          Convert TYPE to character
      CTYPE = 'M'
      IF ( TYPE .EQ. 2 ) CTYPE = 'H'
      IF ( TYPE .EQ. 3 ) CTYPE = 'P'
      IF ( TYPE .EQ. 4 ) CTYPE = 'Q'

C          Print the current status
      WRITE(*,200) CTYPE, OBPMIN, OEPMIN, TMPSCR, YLABEL
 200  FORMAT( 'O', 'Current: ', A1, ' B=', F6.1, ' E=', F6.1,
     1     ' T=', F6.1, ' ', A1 )

      RETURN

      END



      SUBROUTINE GETNUM( CHR, STPTR, LABEL, VALUE, PTR )
C
C
C     GETNUM checks to see if there is a real number in the array CHR,
C     and if there is returns the number in VALUE.  If there is not a
C     real number in CHR, one is input from the terminal, unless LABEL
C     is RETURN, in which case the subroutine is exited without changing
C     VALUE.
C
C     Ver:   1.04   11/ 8/84
C
C

      REAL VALUE

      INTEGER*2 STPTR, PTR, PTR2, PTR3
```

```
      CHARACTER*1 CHR(80)

      CHARACTER*80 LINE

      CHARACTER*17 LABEL


C           Find the first non-blank character
 10   DO 20 PTR = STPTR, 80
 20        IF ( CHR(PTR) .NE. ' ' ) GOTO 40

C           The line was blank, input a new line unless LABEL is RETURN
 30   PTR = 81
      IF ( LABEL .EQ. 'RETURN           ' ) RETURN

      WRITE(*,100) LABEL
 100  FORMAT( ' ', 'Enter the ', A17, \ )
      READ(*, '(A80)' ) LINE

C           Convert the input line into a character vector
      READ( LINE, '(80A1)' ) ( CHR(PTR3), PTR3 = 1, 80 )
      STPTR = 1
      GOTO 10

C           Check for illegal characters until the next blank
 40   STPTR = PTR
      DO 50 PTR = STPTR, 80
           IF ( CHR(PTR) .EQ. ' ' ) GOTO 60
 50        IF ( ( ( CHR(PTR) .LT. '0' ) .AND. ( CHR(PTR) .NE. '.' ) )
     1          .OR. ( CHR(PTR) .GT. '9' ) ) GOTO 30

C           Valid real number, convert from an ASCII array to an ASCII
C           variable
 60   WRITE( LINE, '(80A1)' ) '0', ( CHR(PTR2), PTR2 = STPTR, PTR-1 )

C           This line is needed (for an unknown reason!?!?) to make the
C           following line work
      WRITE( *,200)
 200  FORMAT( '+', ' ' )

C           Convert from an ASCII variable to a real number
      READ( LINE, * ) VALUE

      RETURN

      END
```

E.2 INPUT.FOR

```
$STORAGE:2

C
C
C       INPUT.FOR
C
C       by
C
C       Tim Martin
C
C       11/19/84
C
C       These are the input subroutines for PLOT.FOR
C
C



      SUBROUTINE INPUT
C
C       Inputs the output of the SIMAN model and converts the data
C       into a form that is easily plotable
C
C       Ver  1.03    10/24/84
C

$INCLUDE: 'MODELCOM.FOR'


      WRITE(*,100)
 100  FORMAT( ' ', 'Inputting data', \ )

C       Input and convert the machine utilization and machine
C       commitment data

      CALL INCONM( 'A', MUPA, NUPA, MCPA, NCPA )
      CALL INCONM( 'B', MUPB, NUPB, MCPB, NCPB )
      CALL INCONM( 'C', MUPC, NUPC, MCPC, NCPC )
      CALL INCONM( 'D', MUPD, NUPD, MCPD, NCPD )

C       Input and convert the robot data

      CALL INCONR

C       Input the number of pallets on the conveyor data

      CALL INCNVY

      WRITE(*,*)
```

```
      RETURN
      END



      SUBROUTINE INCONM( MACH, MUP, NUP, MCP, NCP )
C
C       INCONM converts the 'raw' machine utilization and commitment
C       data into the plot data for machine utilization and machine
C       commitment.
C
C       Ver  1.04    11/19/84
C

      REAL MUP(500), MCP(500), MU(1000), MUT(1000), MC(1000), MCT(1000)

      CHARACTER*1 MACH
      CHARACTER*9 FNMU, FNMUT, FNMC, FNMCT

      INTEGER*2 NUP, NCP, NMU, NMC, MACHN, DUMMY


C       Define the file names

      MACHN = (ICHAR( MACH ) - 65)*4 + 76
      WRITE( FNMU, 100 ) MACHN
      WRITE( FNMUT, 100 ) MACHN+1
      WRITE( FNMC, 100 ) MACHN+2
      WRITE( FNMCT, 100 ) MACHN+3
 100  FORMAT( 'OUTPUT.', I2 )

C       Set up the input files

      OPEN( 3, FILE=FNMU, STATUS='OLD' )
      OPEN( 4, FILE=FNMUT, STATUS='OLD' )
      OPEN( 5, FILE=FNMC, STATUS='OLD' )
      OPEN( 6, FILE=FNMCT, STATUS='OLD' )

C       Read the data

      CALL INFILE( 3, MU, NMU )
      CALL INFILE( 4, MUT, DUMMY )
      CALL INFILE( 5, MC, NMC )
      CALL INFILE( 6, MCT, DUMMY )

C       Convert the machine utility data

      CALL CONUTL( MU, MUT, NMU, MUP, NUP )

C       Convert the machine commitment data
```

```
      CALL CONCOM( MC, MCT, NMC, MUP, NUP, MCP, NCP )

C       Close the input files

      CLOSE (3)
      CLOSE (4)
      CLOSE (5)
      CLOSE (6)

      RETURN
      END


      SUBROUTINE CONUTL( MU, MUT, NMU, MUP, NUP )
C
C       CONULT converts the raw machine utilization data into plot
C       data.  The machine is utilized from MUP(k) to MUP(k+1) were
C       k is odd.
C
C       Ver  1.02    11/19/84
C

      REAL MU(500), MUT(1000), MUP(1000)

      INTEGER*2 NMU, NUP, PTRMU, PTRMUT


      PTRMU = 1

C       Find the first utilization period

 10   IF ( MU(PTRMU) .EQ. 1 ) GOTO 20
           PTRMU = PTRMU + 1
           IF ( PTRMU .LT. NMU ) GOTO 10

C       There is no utilized period
      NUP = 0
      RETURN

 20   NUP = 0

C       Take 2, skip 2 to the end of MUT

      DO 30 PTRMUT = PTRMU, NMU, 4

           MUP( NUP+1 ) = MUT( PTRMUT )
           MUP( NUP+2 ) = MUT( PTRMUT + 1 )
 30        NUP = NUP + 2

      RETURN
```

```
      END


      SUBROUTINE CONCOM( MC, MCT, NMC, MUP, NUP, MCP, NCP )
C
C       CONCOM converts the raw machine commitment data and the
C       machine utilization plot data into machine commitment plot
C       data.  For the plot data the machine is not committed when
C       when the machine is utilized.  (This is to prevent two
C       things from being plotted at the same place.)
C
C       Ver  1.04    11/19/84
C

        REAL MC(500), MCT(1000), MUP(1000), MCP(1000)

        INTEGER*2 NMC, NUP, NCP, PTRMC, PTRMUP


        PTRMC = 1

C       Find the first committed period
 10   IF ( MC(PTRMC) .EQ. 1 ) GOTO 20
           PTRMC = PTRMC + 1
           IF ( PTRMC .LT. NMC ) GOTO 10

C       There is no committed period
      NCP = 0
      RETURN

 20   PTRMUP = 1
      NCP = 0

C       Check to see if there are any utilized periods
      IF ( NUP .EQ. 0 ) GOTO 30

C       See if the first committed period is totally disjoint from the
C       first utilized period
      IF ( MCT(PTRMC+1) .LT. MUP(1) ) THEN
           MCP(1) = MCT(PTRMC)
           MCP(2) = MCT(PTRMC+1)
           NCP = 2
           PTRMC = PTRMC + 4
           GOTO 30
      ENDIF

C       Check to see if the first part of the first committed period
C       is also part of the first utilized period
      IF ( MCT(PTRMC) .LT. MUP(1) ) GOTO 30
```

```
C       If it is, the first committed only period follows the utilized
C       period
           MCP(1) = MUP(2)
           MCP(2) = MCT(PTRMC+1)
           PTRMUP = 3
           PTRMC = PTRMC + 4
           NCP = 2

C       Take 2 from MCT, skip 2.  Between the 2 taken from MCT, put
C       2 values from MUP
 30        MCP(NCP+1) = MCT(PTRMC)

C       Check to see if there are any more utilized periods
           IF ( PTRMUP .GT. NUP ) GOTO 40

           MCP(NCP+2) = MUP(PTRMUP)
           MCP(NCP+3) = MUP(PTRMUP+1)
           MCP(NCP+4) = MCT(PTRMC+1)
           PTRMC = PTRMC + 4
           PTRMUP = PTRMUP + 2
           NCP = NCP + 4

      IF ( PTRMC .LT. NMC ) GOTO 30

      IF ( MCP(NCP-1) .EQ. MCP(NCP) ) NCP = NCP - 2

      RETURN

C       If there are no more utilized periods, this has to be the
C       last committed period
 40   NCP = NCP + 2
      MCP(NCP) = MCT(PTRMC+1)

      RETURN

      END


      SUBROUTINE INCONR
C
C
C       INCONR inputs the raw robot utility and position data and
C       converts it to data that will be used to plot where the robot
C       is, and what the status of the robot is.  The robot has three
C       status conditions:  idle, moving without carrying a pallet
C       and moving while carrying a pallet.
C
C       Ver  1.06    11/19/84
C
C
```

```
$INCLUDE: 'MODELCOM.FOR'


      REAL RU(1000), RUT(1000), RP(1000), RPT(1000)

      INTEGER*2 NRU, NRP, DUMMY

C       Open the files to be input
      OPEN( 3, FILE='OUTPUT.92', STATUS='OLD' )
      OPEN( 4, FILE='OUTPUT.93', STATUS='OLD' )
      OPEN( 5, FILE='OUTPUT.96', STATUS='OLD' )
      OPEN( 6, FILE='OUTPUT.97', STATUS='OLD' )

C       Read the data
      CALL INFILE( 3, RU, NRU )
      CALL INFILE( 4, RUT, DUMMY )
      CALL INFILE( 5, RP, NRP )
      CALL INFILE( 6, RPT, DUMMY )

C       Find out when the robot is carrying a pallet
      CALL CONCRY( RU, RUT, RP, RPT, NRP, RCARYT, RCARYP, NRCARY )

C       Find when the robot is idle
      CALL CONIDL( RU, RUT, NRU, RCARYT, RCARYP, RP(1),
     1     RIDLET, RIDLEP, NRIDLE )

C       Find when the robot is moving without a pallet
      CALL CONMOV( RU, RUT, NRU, RCARYT, RCARYP, NRCARY, RP(1),
     1     RMOVET, RMOVEP, NRMOVE )

C       Close the input files
      CLOSE(3)
      CLOSE(4)
      CLOSE(5)
      CLOSE(6)

      RETURN
      END



      SUBROUTINE CONCRY( RU, RUT, RP, RPT, NRP,
     1     RCARYT, RCARYP, NRCARY )
C
C
C       CONCRY converts the input data into a list of robot position
C       and time when the robot is busy carrying a pallet.  The robot
C       always carries a pallet for exactly three line segments. The
C       segments are pick, move and place.  Four data points are needed
C       to represent the three segments.  Therefore the output data
C       is grouped into sets of 4 data points, with each set
```

```
C       representing a complete carry by the robot.  If the first or
C       last carry is not complete, then it is filled with redundant
C       known data to make a set of four data points.
C
C       Ver  1.03    11/19/84
C
C

      REAL RU(1000), RUT(1000), RP(1000), RPT(1000), RCARYT(500)

      INTEGER*2 NRP, RCARYP(500), NRCARY, PTRRU, PTRRP


      PTRRU = 2

C       Find the end of the first carry period
 10   IF ( ( RU(PTRRU-1) .EQ. 1. ) .AND. ( RU(PTRRU) .EQ. 1. ) ) GOTO 20
           PTRRU = PTRRU + 1
           GOTO 10

C       Work backwards from RUT(PTRRU) to get the first carry period
 20   PTRRP = 1

C       Find where in RP the fist carry period ends
 30   IF ( RPT(PTRRP) .GE. RUT(PTRRU) ) GOTO 40
           PTRRP = PTRRP + 1
           GOTO 30

 40   RCARYT(4) = RPT(PTRRP)
      RCARYP(4) = RP(PTRRP)
      RCARYT(3) = RPT(PTRRP-1)
      RCARYP(3) = RP(PTRRP-1)

C       Check to see if the first carry period is complete
      IF ( PTRRP .GE. 6 ) THEN

C       The first period is complete, save the data
           RCARYT(2) = RPT(PTRRP-4)
           RCARYP(2) = RP(PTRRP-4)
           RCARYT(1) = RPT(PTRRP-5)
           RCARYP(1) = RP(PTRRP-5)

        ELSE

C       The first period is not complete, find out how much data is
C       missing and pad as needed (the earliest known data is used
C       for the unknown data
           IF ( PTRRP .GE. 4 ) THEN
                RCARYT(2) = RPT(PTRRP-3)
                RCARYP(2) = RP(PTRRP-3)
             ELSE
```

```
                RCARYT(2) = RCARYT(3)
                RCARYP(2) = RCARYP(3)
            ENDIF
            RCARYT(1) = RCARYT(2)
            RCARYP(1) = RCARYP(2)

        ENDIF

C       Set up for the main loop

        NRCARY = 4
        PTRRP = PTRRP + 8

C
C       Take 2 from RP and RPT, throw 2 away, take 2, throw 2 away
C

C       Check for the end of the input
 50     IF ( PTRRP .GT. NRP ) GOTO 60
            RCARYT(NRCARY+1) = RPT(PTRRP-5)
            RCARYT(NRCARY+2) = RPT(PTRRP-4)
            RCARYT(NRCARY+3) = RPT(PTRRP-1)
            RCARYT(NRCARY+4) = RPT(PTRRP)
            RCARYP(NRCARY+1) = RP(PTRRP-5)
            RCARYP(NRCARY+2) = RP(PTRRP-4)
            RCARYP(NRCARY+3) = RP(PTRRP-1)
            RCARYP(NRCARY+4) = RP(PTRRP)

            NRCARY = NRCARY + 4
            PTRRP = PTRRP + 8
            GOTO 50

C       Find the final values
 60     IF ( ( PTRRP - 4 ) .GT. NRP ) GOTO 70

C       If there is a partial period left, save it and pad with the
C       last known data
            RCARYT(NRCARY+1) = RPT(PTRRP-5)
            RCARYT(NRCARY+2) = RPT(PTRRP-4)
            RCARYT(NRCARY+3) = RCARYT(NRCARY+2)
            RCARYT(NRCARY+4) = RCARYT(NRCARY+2)
            RCARYP(NRCARY+1) = RP(PTRRP-5)
            RCARYP(NRCARY+2) = RP(PTRRP-4)
            RCARYP(NRCARY+3) = RCARYP(NRCARY+2)
            RCARYP(NRCARY+4) = RCARYP(NRCARY+2)

            NRCARY = NRCARY + 4

 70     RETURN

        END
```

```
      SUBROUTINE CONIDL( RU, RUT, NRU, RCARYT, RCARYP, RP1,
     1     RIDLET, RIDLEP, NRIDLE )
C
C
C       CONIDL uses the robot utility data to find when the robot is
C       idle.  The data in RCARY is used to find where the robot was
C       when it was idle.
C
C       Ver  1.03    11/19/84
C
C

      REAL RU(1000), RUT(1000), RCARYT(500), RIDLET(500), RP1

      INTEGER*2 NRU, RCARYP(500), RIDLEP(500), NRIDLE, PTRRU, PTRRCY


      NRIDLE = 0

C       Check to see if the initial position is in RCARY
      IF ( RU(1) .EQ. 0. ) THEN

C       If it is not, use the first value in RP as the initial position
         RIDLET(1) = RUT(1)
         RIDLET(2) = RUT(2)
         RIDLEP(1) = RP1
         RIDLEP(2) = RIDLEP(1)

         NRIDLE = 2

      ENDIF

      PTRRCY = 4

C
C       The main loop.  Go through RU by 2's
C

      DO 30 PTRRU = 3, NRU, 2

C       Check to see if the robot is idle
         IF ( RU(PTRRU) .EQ. 0. ) THEN

              RIDLET(NRIDLE+1) = RUT(PTRRU)
              RIDLET(NRIDLE+2) = RUT(PTRRU+1)

C       Find the position the robot was in when it was idle
 10           IF ( RCARYT(PTRRCY) .GE. RIDLET(NRIDLE+1) ) GOTO 20
```

```
                    PTRRCY = PTRRCY + 4
                    GOTO 10

 20             RIDLEP(NRIDLE+1) = RCARYP(PTRRCY)
                RIDLEP(NRIDLE+2) = RIDLEP(NRIDLE+1)

                NRIDLE = NRIDLE + 2

           ENDIF
 30        CONTINUE

      RETURN

      END



      SUBROUTINE CONMOV( RU, RUT, NRU, RCARYT, RCARYP, NRCARY, RP1,
     1     RMOVET, RMOVEP, NRMOVE )
C
C
C       CONMOV finds when the robot is moving without a pallet.
C       This is only occurs before a robot picks up a pallet at a
C       station different from the station it last left a pallet at.
C       The move data is stored in RMOVE in pairs, with the odd values
C       the start of a move and the even values the end of a move.
C       (The move is always just a line segment.)
C
C       Ver  2.03    11/19/84
C
C

      REAL RU(1000), RUT(1000), RCARYT(500), RMOVET(500), RP1

      INTEGER*2 RCARYP(500), RMOVEP(500), NRMOVE, PTRRU, PTRRCY, NRCARY,
     1     NRU


C
C       Check the initial condition
C

      NRMOVE = 0
      PTRRU = 1

C       Find when the robot is first busy
 10   IF ( RU(PTRRU) .EQ. 1. ) GOTO 20
           PTRRU = PTRRU + 2
           GOTO 10

C       See if the robot was busy before the first carry period
```

```
 20   IF ( RUT(PTRRU) .LT. RCARYT(1) ) THEN

C       It was, so it moved without a pallet before the first carry
           RMOVET(1) = RUT(PTRRU)
           RMOVET(2) = RCARYT(1)
           RMOVEP(1) = RP1
           RMOVEP(2) = RCARYP(1)

           NRMOVE = 2

      ENDIF

C
C       The main program loop - go through RCARYP
C

      DO 50 PTRRCY = 4, NRCARY-4, 4

C       See if the robot moved between carry periods
           IF ( RCARYP(PTRRCY) .NE. RCARYP(PTRRCY+1) ) THEN

C       It did, so find when it started to move
 30              IF ( RUT(PTRRU+1) .GE. RCARYT(PTRRCY+1) ) GOTO 40
                      PTRRU = PTRRU + 2
                      GOTO 30

C       Save the move data
 40              RMOVET(NRMOVE+1) = RUT(PTRRU)
                 RMOVET(NRMOVE+2) = RCARYT(PTRRCY+1)
                 RMOVEP(NRMOVE+1) = RCARYP(PTRRCY)
                 RMOVEP(NRMOVE+2) = RCARYP(PTRRCY+1)

                 NRMOVE = NRMOVE + 2

           ENDIF
 50        CONTINUE

C
C       The final condition - check to see if the robot is busy after
C       the last carry period
C

      IF ( (RU(NRU-1) .EQ. 1. ) .AND.
     1      ( RUT(NRU-1) .GE. RCARYT(NRCARY) ) ) THEN
C       The robot is moving without a pallet at the end of the run
           RMOVET(NRMOVE+1) = RUT(NRU-1)
           RMOVET(NRMOVE+2) = RUT(NRU)
           RMOVEP(NRMOVE+1) = RCARYP(NRCARY)
           RMOVEP(NRMOVE+2) = RMOVEP(NRMOVE+1)

           NRMOVE = NRMOVE + 2
```

```
      ENDIF

      RETURN

      END



      SUBROUTINE INCNVY
C
C       INCNVY inputs the number of pallets that were on the input
C       conveyor at each time in the run
C
C       Ver  1.03    11/19/84
C

$INCLUDE: 'MODELCOM.FOR'

      REAL CONINN(1000), CONINT(1000)

      INTEGER*2 PTR, DUMMY, NCONIN


C       Open the disk files
      OPEN( 3, FILE='OUTPUT.94', STATUS='OLD' )
      OPEN( 4, FILE='OUTPUT.95', STATUS='OLD' )

C       Read in the data
      CALL INFILE( 3, CONINN, NCONIN )
      CALL INFILE( 4, CONINT, DUMMY )

C       Remove all of the even data points
      NCONVY = 1
      DO 10 PTR = 1, NCONIN, 2
           CONVYN(NCONVY) = CONINN(PTR)
           CONVYT(NCONVY) = CONINT(PTR)
 10        NCONVY = NCONVY + 1

C       Except for the last value
      CONVYN(NCONVY) = CONINN(NCONIN)
      CONVYT(NCONVY) = CONINT(NCONIN)

C       Close the input files
      CLOSE (3)
      CLOSE (4)

      RETURN

      END
```

```
      SUBROUTINE INFILE( UNIT, ARRAY, LEN )
C
C       INFILE inputs the real array stored in unit UNIT.  The first
C       two elements and the last element in the file is not saved.
C       (The first 3 elements are identical and the last 2 elements
C       are the same.)  The data is saved in the array ARRAY, which
C       has a length LEN.  The maximum length of ARRAY is 1000.
C
C       Ver  1.03    11/ 8/84
C

      INTEGER*2 UNIT, LEN

      REAL TRASH, ARRAY(*)


C       Throw away the first two values

      READ( UNIT, * ) TRASH
      READ( UNIT, * ) TRASH

      DO 10 LEN = 1, 1001
 10        READ( UNIT, *, END=20 ) ARRAY(LEN)

      WRITE( *, * ) 'MORE THAN 1000 VALUES INPUT!'
      RETURN

C       Throw away the last value and adjust LEN for the time
C       thru the loop when the end of file was reached

 20   LEN = LEN - 2

      WRITE(*,100)
 100  FORMAT( '.', \ )

      RETURN
      END
```

E.3 SCREEN.FOR

```
$STORAGE:2
C
C
C   SCREEN.FOR
C
C   Contains the routines for setting up the screen for plotting
C
C   11/ 6/84
C
C


      SUBROUTINE SCREEN
C
C
C     SCREEN puts a plot on the screen
C
C     Ver:   1.11   10/30/84
C
C

$INCLUDE: 'MODELCOM.FOR'
$INCLUDE: 'PLOTCOM.FOR'

      REAL YMIN, YMAX, YOVERX, ASPECT, TIMTIC, YTICS, TICMOD, TIMNED

      INTEGER*2 BCKGND, CLRPAL, ROW1, ROW2, SCALE, LTICT, COL1, COL2,
     1     LAXIS, NDEC, LTICY, NDOTS, SYMBOL, CLRMU, CLRMC, CLRRCY,
     2     CLRRID, CLRRMV, BASE, MODE, CLRCNV


C          Clear the screen, set the mode and color pallet
      MODE = 4
      IF ( TYPE .EQ. 2 ) MODE = 6
      CALL QSMODE( MODE )
      BCKGND = 0
      IF ( MODE .EQ. 6 ) BCKGND = 7
      CLRPAL = 0
      CALL QCOLOR( BCKGND, CLRPAL )

C          Set up the plot area
      COL1 = 64
      IF ( YLABEL .EQ. 'N' ) COL1 = 0
      COL2 = 318
      IF ( MODE .EQ. 6 ) COL2 = 638
      ROW1 = 17
      ROW2 = 191
      YMIN = 0
      YMAX = 7
      SCALE = 0
```

```
      YOVERX = 1
      ASPECT = 1

C           Find the major tic mark intervals
      TIMNED = (8*4/60.) * ( EPTIME - BPTIME ) / ( COL2 - COL1 + 1 )
      BASE = 10 ** INT( ALOG10( TIMNED ) )
      TIMTIC = BASE * NINT( TIMNED / BASE + .4 )

C           Find the first time that is exactly divisible by TIMTIC
      TICMOD = AINT( (BPTIME/60) / TIMTIC ) * TIMTIC

      CALL QPLOT( COL1, COL2, ROW1, ROW2, BPTIME, EPTIME, YMIN, YMAX,
     1     TICMOD*60, YMIN, SCALE, YOVERX, ASPECT )

C           Draw the axes
      LTICT = 1
      LAXIS = 0
      NDEC = 0
      CALL QXAXIS( BPTIME, EPTIME, TIMTIC*60, LTICT, LAXIS, NDEC )

C           Reset the y-axis to BPTIME
      CALL QPLOT( COL1, COL2, ROW1, ROW2, BPTIME, EPTIME, YMIN, YMAX,
     1     BPTIME, YMIN, SCALE, YOVERX, ASPECT )

      YTICS = 0
      LTICY = 0
      IF ( YLABEL .EQ. 'Y' ) CALL QYAXIS( YMIN, YMAX, YTICS, LTICY,
     1     LAXIS, NDEC )

C           Label the axes
      CALL SLBAXE( TIMTIC, COL1, COL2, TICMOD )

C           Draw when the robot was idle
      IF ( MODE .EQ. 4 ) THEN
           NDOTS = 0
           CLRRID = 1
      ENDIF
      IF ( MODE .EQ. 6 ) THEN
           NDOTS = 4
           CLRRID = 3
      ENDIF
      SYMBOL = -2
      CALL QSETUP( NDOTS, CLRRID, SYMBOL, CLRRID )

      CALL PRBT( RIDLET, RIDLEP, NRIDLE, 'S' )

C           Draw when the robot was moving without a pallet
      CLRRMV = 3
      IF (MODE .EQ. 6 ) NDOTS = 6
      CALL QSETUP( NDOTS, CLRRMV, SYMBOL, CLRRMV )
```

```
      CALL PRBT( RMOVET, RMOVEP, NRMOVE, 'S' )

C           Draw the robot carry periods
      CLRRCY = 2
      IF ( MODE .EQ. 6 ) THEN
           NDOTS = 0
           CLRRCY = 3
      ENDIF
      CALL QSETUP( NDOTS, CLRRCY, SYMBOL, CLRRCY )

      CALL PRCY( 'S' )

C           Draw the machine commitment data
      CLRMC = 3
      IF ( MODE .EQ. 6 ) NDOTS = 4
      CALL QSETUP( NDOTS, CLRMC, SYMBOL, CLRMC )

      CALL PMACH( MCPA, NCPA, 2.12, 'S' )
      CALL PMACH( MCPB, NCPB, 3.12, 'S' )
      CALL PMACH( MCPC, NCPC, 4.12, 'S' )
      CALL PMACH( MCPD, NCPD, 5.12, 'S' )

C           Draw the machine utility data
      CLRMU = 2
      IF ( MODE .EQ. 6 ) THEN
           NDOTS = 0
           CLRMU = 3
      ENDIF
      CALL QSETUP( NDOTS, CLRMU, SYMBOL, CLRMU )

      CALL PMACH( MUPA, NUPA, 2.12, 'S' )
      CALL PMACH( MUPB, NUPB, 3.12, 'S' )
      CALL PMACH( MUPC, NUPC, 4.12, 'S' )
      CALL PMACH( MUPD, NUPD, 5.12, 'S' )

C           Put the number of pallets waiting on the input conveyor
C           on the plot
      CLRCNV = 3
      NDOTS = 0
      CALL QSETUP( NDOTS, CLRCNV, SYMBOL, CLRCNV )

      CALL PCONVY( 'S' )

C           Move the cursor to the upper left corner of the screen
      CALL QCMOV( 0, 25 )

      RETURN

      END
```

```
      SUBROUTINE SLBAXE( TIMTIC, COL1, COL2, TICMOD )
C
C
C     SLBAXE labels the time and machine axes
C
C     Ver:   1.09   11/ 6/84
C
C

$INCLUDE: 'PLOTCOM.FOR'

      REAL TIMTIC, TICMOD

      INTEGER*2 NTIC, NCHAR, TMTICS(30), ROW(30), PTRTIC, FSTTIC, COL1,
     1     COL2, TIME, PIX, TXTCOL, MISCHR, TMTICO

      CHARACTER*4 CTIME

      CHARACTER*24 TITLE

C          Define function NCHAR:  Find out how many digits in IT
      NCHAR(IT) = ALOG10( FLOAT(IT) + .1 ) + 1


C
C          Put the title on the plot
C

      WRITE( TITLE ,200) BPTIME/60., EPTIME/60.
 200  FORMAT( 'Plot from', F6.1, ' to', F6.1 )
      CALL QPTXT( 24, TITLE, 3, COL2/2/8-12, 24 )

C
C          Label the y-axis
C

      IF ( YLABEL .EQ. 'N' ) GOTO 30
      CALL QCMOV( 0, 22 )
      WRITE(*, 100)
 100  FORMAT( ' ', ' Output', /, ' ', 'Conveyor', /, '0', 'Machine', /,
     1     ' ', '   D', /, '0', 'Machine', /, ' ', '   C', /, '0',
     2     'Machine', /, ' ', '   B', /, '0', 'Machine', /, ' ',
     3     '   A', /, /, '0', ' Input', /, ' ', 'Conveyor' )

C
C          Label the time axis
C

C          Find the pixel locations of the major tic marks
 30   CALL QXTICS( NTIC, TMTICS, ROW )
```

```
C          Find the last time label
      TIME = TICMOD + TIMTIC * NTIC

C          Check to see if the last label can be printed
      MISCHR = ( TMTICS(NTIC) + 4*NCHAR(TIME) - COL2 + 7 ) / 8

C          If there are missing characters, don't print the label
      IF ( MISCHR .GE. 1 ) GOTO 20

C          Find the start pixel column for the label
      PIX = TMTICS(NTIC) - 4*NCHAR(TIME) -
     1      8 * (  4 - NCHAR(TIME) )

C          Convert the time to a character string
      WRITE( CTIME, '(I4)' ) TIME

C          Put the time on the screen
      CALL QGTXT( 4, CTIME, 3, PIX, 8, 0 )

C          Decrement the time
 20   TIME = TIME - TIMTIC

C          Put the time labels on in reverse order
      DO 10 PTRTIC = NTIC-1, 1, -1

C               Find the start pixel column for the label
           PIX = TMTICS(PTRTIC) - 4*NCHAR(TIME) - 8*( 4 - NCHAR(TIME) )

C               Convert the time to a character string
           WRITE( CTIME, '(I4)' ) TIME

C               Put the time on the screen
           CALL QGTXT( 4, CTIME, 3, PIX, 8, 0 )

C               Decrement to the next time
 10        TIME = TIME - TIMTIC

C          Check to see if there is a tic mark at the origin
      TMTIC0 = TMTICS(1) - ( TMTICS(2) - TMTICS(1) )
      IF ( TMTIC0 .GE. COL1-4 ) THEN

C               There is a tic at the origin. Print the label
           IF ( TIME .LE. 0 ) THEN
                PIX = TMTIC0 - 28
              ELSE
                PIX = TMTIC0 - 4*NCHAR(TIME) - 8*( 4 - NCHAR(TIME) )
           ENDIF

C               If the label fits on the screen, print it
           IF ( PIX .GE. 0 ) THEN
```

```
            WRITE( CTIME, '(I4)' ) TIME
            CALL QGTXT( 4, CTIME, 3, PIX, 8, 0 )
         ENDIF

      ENDIF

C           Put the 'Time' label on the screen
      TXTCOL = ( ( COL2 - COL1 ) / 2 + COL1 ) / 8  -  5
      CALL QPTXT( 10, 'Time (Min)', 3, TXTCOL, 0 )


      RETURN

      END
```

E.4 PLOTTER.FOR

```
$STORAGE:2
C
C
C   PLOTTER.FOR
C
C   Contains the subroutines for setting up a plot on the pen plotter
C
C   11/ 6/84
C
C


      SUBROUTINE PLOTER
C
C
C     PLOTER puts a plot on the IBM 749 pen plotter
C
C     Ver:   1.04   11/ 5/84
C
C

$INCLUDE: 'MODELCOM.FOR'
$INCLUDE: 'PLOTCOM.FOR'

      REAL YMIN, YMAX, TIMTIC, YTICS, TICMOD, TIMNED

      INTEGER*2 ROW1, ROW2, LTICT, COL1, COL2, LINTYP, TTLHGT, TTLWDT,
     1     LAXIS, NDEC, LTICY, SYMBOL, CLRMU, CLRMC, CLRRCY,
     2     CLRRID, CLRRMV, BASE, CLRCNV, LBLHGT, LBLWDT, TMLHGT, TMLWDT


C          Initialize the plot, set the character sizes
      CALL ZINIT( 1, 1, 'DUMMYF.ILE' )
      LBLHGT = 60
      LBLWDT = 36
      TMLHGT = 42
      TMLWDT = 25
      TTLHGT = 84
      TTLWDT = 51

C          Set up the plot area
      COL1 = 8 * LBLWDT + 76
      IF ( YLABEL .EQ. 'N' ) COL1 = 2*TMLWDT + 76
      COL2 = 2690 - 2*TMLWDT - 76
      ROW1 = 1.25*LBLHGT + 1.125 * TMLHGT + 76
      ROW2 = 2060 - 1.25*TTLHGT - 76
      YMIN = 0
      YMAX = 6.5

C          Find the major tic mark intervals
```

```
      TIMNED = (TMLWDT*4/60.) * ( EPTIME - BPTIME ) / (COL2 - COL1 + 1)
      BASE = 10 ** INT( ALOG10( TIMNED ) )
      TIMTIC = BASE * NINT( TIMNED / BASE + .4 )

C           Find the first time that is exactly divisible by TIMTIC
      TICMOD = AINT( (BPTIME/60) / TIMTIC ) * TIMTIC

      CALL ZPLOT( COL1, COL2, ROW1, ROW2, BPTIME, EPTIME, YMIN, YMAX,
     1     TICMOD*60, YMIN )

C           Draw the axes
      LTICT = 1
      LAXIS = 0
      NDEC = 0
      CALL ZXAXIS( BPTIME, EPTIME, TIMTIC*60, LTICT, LAXIS, NDEC )

C           Reset the y-axis to BPTIME
      CALL ZPLOT( COL1, COL2, ROW1, ROW2, BPTIME, EPTIME, YMIN, YMAX,
     1     BPTIME, YMIN )

      YTICS = 0
      LTICY = 0
      IF ( YLABEL .EQ. 'Y' ) CALL ZYAXIS( YMIN, YMAX, YTICS, LTICY,
     1     LAXIS, NDEC )

C           Label the axes
      CALL PLBAXE( TIMTIC, COL1, COL2, TICMOD, LBLHGT, LBLWDT, TMLHGT,
     1     TMLWDT, ROW1, ROW2, TTLHGT, TTLWDT )

C           Draw when the robot was idle
      LINTYP = -1
      SYMBOL = 0
      CLRRID = 3
      CALL ZSETUP( LINTYP, CLRRID, SYMBOL, CLRRID )

      CALL PRBT( RIDLET, RIDLEP, NRIDLE, 'P' )

C           Draw when the robot was moving without a pallet
      CLRRMV = 4
      CALL ZSETUP( LINTYP, CLRRMV, SYMBOL, CLRRMV )

      CALL PRBT( RMOVET, RMOVEP, NRMOVE, 'P' )

C           Draw the robot carry periods
      CLRRCY = 5
      CALL ZSETUP( LINTYP, CLRRCY, SYMBOL, CLRRCY )

      CALL PRCY( 'P' )

C           Draw the machine commitment data
      CLRMC =  6
```

```
      CALL ZSETUP( LINTYP, CLRMC, SYMBOL, CLRMC )

      CALL PMACH( MCPA, NCPA, 2.12, 'P' )
      CALL PMACH( MCPB, NCPB, 3.12, 'P' )
      CALL PMACH( MCPC, NCPC, 4.12, 'P' )
      CALL PMACH( MCPD, NCPD, 5.12, 'P' )

C          Draw the machine utility data
      CLRMU = 7
      CALL ZSETUP( LINTYP, CLRMU, SYMBOL, CLRMU )

      CALL PMACH( MUPA, NUPA, 2.12, 'P' )
      CALL PMACH( MUPB, NUPB, 3.12, 'P' )
      CALL PMACH( MUPC, NUPC, 4.12, 'P' )
      CALL PMACH( MUPD, NUPD, 5.12, 'P' )

C          Put the number of pallets waiting on the input conveyor
C          on the plot
      CLRCNV = 8
      CALL ZSETUP( LINTYP, CLRCNV, SYMBOL, CLRCNV )

      CALL PCONVY( 'P' )

C          Wrap up the plot
      CALL ZFINIS

      RETURN

      END



      SUBROUTINE PLBAXE( TIMTIC, COL1, COL2, TICMOD, LBLHGT, LBLWDT,
     1     TMLHGT, TMLWDT, ROW1, ROW2, TTLHGT, TTLWDT )
C
C
C     PLBAXE labels the time and machine axes
C
C     Ver:   1.05   11/ 6/84
C
C

$INCLUDE: 'PLOTCOM.FOR'

      REAL TIMTIC, TICMOD

      INTEGER*2 NTIC, NCHAR, TMTICS(30), ROW(30), PTRTIC, FSTTIC, COL1,
     1     COL2, TIME, PIX, MISCHR, TMTICO, LBLHGT, LBLWDT, TMHPIX,
     2     TMLHGT, TMLWDT, ROW1, ROW2, TTLHGT, TTLWDT, R2M1D7, LHD16,
     3     LEFT
```

```
      CHARACTER*4 CTIME

      CHARACTER*24 TITLE

C           Define function NCHAR:  Find out how many digits in IT
      NCHAR(IT) = ALOG10( FLOAT(IT) + .1 ) + 1


C           Set the plot window to the entire page
      CALL ZCLIP( 0, 2690, 0, 2060 )

C
C           Label the y-axis
C

      IF ( YLABEL .EQ. 'N' ) GOTO 30
C           Set the character size
      CALL ZCHSIZ( LBLHGT, 0, LBLWDT )

C           Put the labels on
      R2M1D7 = ( ROW2 - ROW1 ) / 6.5
      LBLHD8 = LBLHGT/8
      CALL ZPTXT( 6, 'Output', 1,
     1     COL1 - 7*LBLWDT, 6*R2M1D7+ROW1+LBLHD8 )
      CALL ZPTXT( 8, 'Conveyor', 1,
     1     COL1 - 8*LBLWDT, 6*R2M1D7+ROW1-9*LBLHD8 )
      CALL ZPTXT( 7, 'Machine', 1,
     1     COL1 - 8*LBLWDT, INT(5.06*R2M1D7+ROW1+LBLHD8) )
      CALL ZPTXT( 1, 'D', 1,
     1     COL1 - 5*LBLWDT, INT(5.06*R2M1D7+ROW1-9*LBLHD8) )
      CALL ZPTXT( 7, 'Machine', 1,
     1     COL1 - 8*LBLWDT, INT(4.06*R2M1D7+ROW1+LBLHD8) )
      CALL ZPTXT( 1, 'C', 1,
     1     COL1 - 5*LBLWDT, INT(4.06*R2M1D7+ROW1-9*LBLHD8) )
      CALL ZPTXT( 7, 'Machine', 1,
     1     COL1 - 8*LBLWDT, INT(3.06*R2M1D7+ROW1+LBLHD8) )
      CALL ZPTXT( 1, 'B', 1,
     1     COL1 - 5*LBLWDT, INT(3.06*R2M1D7+ROW1-9*LBLHD8) )
      CALL ZPTXT( 7, 'Machine', 1,
     1     COL1 - 8*LBLWDT, INT(2.06*R2M1D7+ROW1+LBLHD8) )
      CALL ZPTXT( 1, 'A', 1,
     1     COL1 - 5*LBLWDT, INT(2.06*R2M1D7+ROW1-9*LBLHD8) )
      CALL ZPTXT( 5, 'Input', 1,
     1     COL1 - 7*LBLWDT, R2M1D7+ROW1+LBLHD8 )
      CALL ZPTXT( 8, 'Conveyor', 1,
     1     COL1 - 8*LBLWDT, R2M1D7+ROW1-9*LBLHD8 )

C
C           Label the time axis
C
```

```
C          Set the time label character size
 30   CALL ZCHSIZ( TMLHGT, 0, TMLWDT )

C          Find the pixel locations of the major tic marks
      CALL ZXTICS( NTIC, TMTICS, ROW )

C          Find the last time label
      TIME = TICMOD + TIMTIC * NTIC
      TMHPIX = ROW1 - 1.125*TMLHGT

C          Put the time labels on in reverse order
      DO 10 PTRTIC = NTIC, 1, -1

C               Find the start pixel column for the label
           PIX = TMTICS(PTRTIC) - TMLWDT/2*NCHAR(TIME) -
     1           TMLWDT*( 4 - NCHAR(TIME) )

C               Convert the time to a character string
           WRITE( CTIME, '(I4)' ) TIME

C               Put the time on the screen
           CALL ZPTXT( 4, CTIME, 1, PIX, TMHPIX )

C               Decrement to the next time
 10        TIME = TIME - TIMTIC

C          Check to see if there is a tic mark at the origin
      TMTIC0 = TMTICS(1) - ( TMTICS(2) - TMTICS(1) )
      IF ( TMTIC0 .GE. COL1-TMLWDT/2 ) THEN

C               There is a tic at the origin. Print the label
           IF ( TIME .LE. 0 ) THEN
                PIX = TMTIC0 - 3.5*TMLWDT
              ELSE
                PIX = TMTIC0 - TMLWDT/2*NCHAR(TIME) -
     1               TMLWDT*( 4 - NCHAR(TIME) )
           ENDIF
           WRITE( CTIME, '(I4)' ) TIME
           CALL ZPTXT( 4, CTIME, 1, PIX, TMHPIX )

      ENDIF

C          Put the 'Time' label on the screen

C          Set the character size
      CALL ZCHSIZ( LBLHGT, 0, LBLWDT )

C          Find where to put the label
      PIX = ( ( COL2 - COL1 ) / 2 + COL1 ) - 5 * LBLWDT
      CALL ZPTXT( 10, 'Time (Min)', 1,
     1     PIX, INT( ROW1-1.25*LBLHGT-1.125*TMLHGT ) )
```

```
C
C          Put the title on
C

C          Set the character size
      CALL ZCHSIZ( TTLHGT, 0, TTLWDT )

C          Create the title
      WRITE( TITLE, 100 ) BPTIME/60., EPTIME/60.
 100  FORMAT( 'Plot from', F6.1, ' to', F6.1 )

C          Put the title on the plot
      LEFT = COL1 - 8*LBLWDT
      PIX = ( COL2 - LEFT ) / 2  + LEFT  -  12*TTLWDT
      CALL ZPTXT( 24, TITLE, 2,
     1     PIX, INT( ROW2+0.25*TTLHGT ) )


      RETURN

      END
```

E.5 MACH.FOR

```
$STORAGE:2
C
C
C   MACH.FOR
C
C   Contains the plotting routine for the machines and conveyor
C
C   11/19/84
C
C

      SUBROUTINE PMACH( MP, NP, POSITN, PTYPE )
C
C
C     PMACH plots machine utility or machine commitment data
C
C     Ver:   1.02   11/19/84
C
C

$INCLUDE: 'PLOTCOM.FOR'

      REAL MP(500)

      INTEGER*2 NP, POSITN, PTRMP

      CHARACTER*1 PTYPE

C          Do for all data points
      DO 10 PTRMP = 1, NP, 2

C              Check to see if the current data point set is in the
C              plot window
           IF ( ( MP(PTRMP+1) .GE. BPTIME ) .AND.
     1          ( MP(PTRMP) .LE. EPTIME ) ) THEN

C                  At least part of the points are plotable
               IF ( ( MP(PTRMP) .GE. BPTIME ) .AND.
     1              ( MP(PTRMP+1) .LE. EPTIME ) ) THEN

C                      The data set is fully plotable
                   CALL PTWO( MP(PTRMP), MP(PTRMP+1), POSITN, POSITN,
     1                      PTYPE )

               ELSE

C                      Find how the data is partially plotable
                   IF ( MP(PTRMP) .LT. BPTIME ) THEN
```

```
C                              The data is cut on the left
                          CALL PTWO( BPTIME, MP(PTRMP+1),
     1                         POSITN, POSITN, PTYPE )

                     ELSE

C                              The data is cut on the right
                          CALL PTWO( MP(PTRMP), EPTIME,
     1                         POSITN, POSITN, PTYPE )

                     ENDIF

                ENDIF

            ENDIF
 10         CONTINUE

      RETURN

      END



      SUBROUTINE PCONVY( PTYPE )
C
C
C     PCONVY plots the number of pallets waiting on the input conveyor
C     as a function of time.  The number is plotted as bars on the plot.
C
C     Ver:   1.03    10/30/84
C
C

$INCLUDE: 'MODELCOM.FOR'
$INCLUDE: 'PLOTCOM.FOR'

      CHARACTER*1 PTYPE

      REAL POSITN, BSEGT, ESEGT

      INTEGER*2 NPLOT, PTR

      LOGICAL*2 PLTSEG, PLTPND


C          Set up for the plot of when there is at least one pallet
C          on the conveyor
      POSITN = 0.88
      NPLOT = 1
      PLTPND = .FALSE.
```

```
C
C               Loop until the maximum number of pallets waiting is
C               reached.  This is indicated by reaching a value of
C               NPLOT for which no segments are plotted
C
 10         PLTSEG = .FALSE.

C               For each value of NPLOT go through CONVY to look
C               for plotable segments
            DO 20 PTR = 1, INT(NCONVY)-1

C                   Check to see if there is a plot pending
                IF ( PLTPND ) THEN

C                       There is, check to see if this is the end of
C                       the segment
                    IF ( ( CONVYN(PTR) .LT. NPLOT ) .OR.
     1                  ( CONVYT(PTR) .GE. EPTIME ) ) THEN

C                           It is, plot the segment
                        ESEGT = AMIN1( CONVYT(PTR), EPTIME )
                        CALL PTWO( BSEGT, ESEGT, POSITN,POSITN, PTYPE)
                        PLTSEG = .TRUE.
                        PLTPND = .FALSE.

                    ENDIF

                ELSE

C                       Check to see if the next segment is plotable
                    IF ( ( CONVYN(PTR) .GE. NPLOT ) .AND.
     1                  ( CONVYT(PTR) .LT. EPTIME ) .AND.
     2                  ( CONVYT(PTR+1) .GT. BPTIME ) ) THEN

C                           It is, set up the beginning of the next
C                           plot
                        PLTPND = .TRUE.
                        BSEGT = AMAX1( CONVYT(PTR), BPTIME )

                    ENDIF

                ENDIF
 20             CONTINUE

C               Check for unfinished plots
            IF ( PLTPND ) THEN

C                   There is a plot pending, plot it using the last
C                   value in CONVYT as the end segment time
                CALL PTWO( BSEGT, CONVYT(PTR), POSITN, POSITN, PTYPE )
                PLTPND = .FALSE.
```

```
                PLTSEG = .TRUE.

            ENDIF

C               Check to see if nothing was plotted
            IF ( .NOT. PLTSEG ) RETURN

C               If something was, increment to the next number of
C               pallets waiting on the input conveyor
            NPLOT = NPLOT + 1
            POSITN = POSITN - 0.08
            GOTO 10

        END



        SUBROUTINE PTWO( T1, T2, Y1, Y2, PTYPE )
C
C
C       PTWO draws a line segment.  The line is drawn from (T1,Y1) to
C       (T2,Y2).
C
C       Ver:    1.01    10/23/84
C
C

        REAL T1, T2, Y1, Y2, T(2), Y(2)

        CHARACTER*1 PTYPE


C           Set up arrays of the input data
        T(1) = T1
        T(2) = T2
        Y(1) = Y1
        Y(2) = Y2

C           Draw the line
        IF ( PTYPE .EQ. 'S' ) CALL QTABL( 1, 2, T, Y )
        IF ( PTYPE .EQ. 'P' ) CALL ZTABL( 1, 2, T, Y )

        RETURN

        END


        SUBROUTINE PFOUR( T1, T2, T3, T4, Y1, Y2, Y3, Y4, PTYPE )
C
C
C       PFOUR draws three continuous line segments.
```

```
C
C     Ver:   1.01   10/23/84
C
C

      REAL T1, T2, T3, T4, Y1, Y2, Y3, Y4, T(4), Y(4)

      CHARACTER*1 PTYPE


C           Convert the input data into an array
      T(1) = T1
      T(2) = T2
      T(3) = T3
      T(4) = T4
      Y(1) = Y1
      Y(2) = Y2
      Y(3) = Y3
      Y(4) = Y4

C           Draw the data
      IF ( PTYPE .EQ. 'S' ) CALL QTABL( 1, 4, T, Y )
      IF ( PTYPE .EQ. 'P' ) CALL ZTABL( 1, 4, T, Y )

      RETURN

      END
```

E.6 ROBOT.FOR

```
$STORAGE:2

C
C
C   ROBOT.FOR
C
C   Contains the plotting routines for the robot
C
C   10/30/84
C
C

      SUBROUTINE PRBT( RT, RP, NR, PTYPE )
C
C
C     PRBT plots when the robot is idle or when the
C     robot is moving without a pallet.  These plots are all just
C     line segments.
C
C     Ver:   1.03   10/16/84
C
C

$INCLUDE: 'PLOTCOM.FOR'

      REAL RT(500), POSITN

      INTEGER*2 RP(500), NR, PTR

      CHARACTER*1 PTYPE


C
C          Find the first plotable segment
C
      PTR = 1

 10   IF ( RT(PTR+1) .GT. BPTIME ) THEN

C               The first plotable segment
           IF ( RT(PTR) .LT. BPTIME ) THEN

C                    Not fully plotable, interpolate the cutoff position
                POSITN = ( BPTIME - RT(PTR) )*( FLOAT( RP(PTR+1) ) -
     1                    FLOAT( RP(PTR) ) ) / ( RT(PTR+1) - RT(PTR) )
     2                      + RP(PTR)
                CALL PTWO( BPTIME, RT(PTR+1), POSITN, FLOAT(RP(PTR+1)),
     1                PTYPE )
                PTR = PTR + 2
```

```
C                    Check for the end of the data
                  IF ( PTR .GE. NR ) RETURN

            ENDIF

      ELSE

C               Increment to the next data pair
            PTR = PTR + 2
            IF ( PTR .GE. NR ) RETURN
            GOTO 10

      ENDIF

C
C           The main plotting loop
C

C           Check for the end of the plotable segments
 20   IF ( RT(PTR+1) .LE. EPTIME ) THEN

C               The next data pair is fully plotable
            CALL PTWO( RT(PTR), RT(PTR+1), FLOAT( RP(PTR) ),
     1           FLOAT( RP(PTR+1) ), PTYPE )
            PTR = PTR + 2

C               Check for the end of the input data
            IF ( PTR .GE. NR ) RETURN
            GOTO 20

      ENDIF

C
C           Check to see if there is a partial final plot
C
      IF ( RT(PTR) .LT. EPTIME ) THEN

C               There is a partial plot, interpolate the cutoff position
            POSITN = ( EPTIME - RT(PTR) ) * ( FLOAT( RP(PTR+1) ) -
     1               FLOAT( RP(PTR) ) ) / ( RT(PTR+1) - RT(PTR) )
     2                 + RP(PTR)
            CALL PTWO( RT(PTR), EPTIME, FLOAT( RP(PTR) ), POSITN, PTYPE )

      ENDIF

      RETURN

      END
```

```
      SUBROUTINE PRCY( PTYPE )
C
C
C     PRCY plots when the robot was carrying a pallet
C
C     Ver:   1.00  10/ 8/84
C
C

$INCLUDE: 'MODELCOM.FOR'
$INCLUDE: 'PLOTCOM.FOR'

      REAL POSITN

      INTEGER*2 PTR

      CHARACTER*1 PTYPE


C
C          Find the fist plotable period
C
      PTR = 1

 10   IF ( RCARYT(PTR+3) .GT. BPTIME ) THEN

C               The first plotable period
            IF ( RCARYT(PTR) .LT. BPTIME ) THEN

C                    Not fully plotable
                  IF ( RCARYT(PTR+1) .GE. BPTIME ) THEN

C                         Only the pick is cutoff
                     CALL PFOUR( BPTIME, RCARYT(PTR+1), RCARYT(PTR+2),
     1                    RCARYT(PTR+3), FLOAT( RCARYP(PTR) ),
     2                    FLOAT( RCARYP(PTR+1) ), FLOAT( RCARYP(PTR+2)),
     3                    FLOAT( RCARYP(PTR+3) ), PTYPE )

                  ELSE

                     IF ( RCARYT(PTR+2) .GT. BPTIME ) THEN

C                            Cut in the middle of the move,
C                            interpolate the cutoff position
                        POSITN = (BPTIME - RCARYT(PTR+1))*
     1                       (RCARYP(PTR+2) - RCARYP(PTR+1)) /
     2                       (RCARYT(PTR+2) - RCARYT(PTR+1))
     3                         +  RCARYP(PTR+1)
                        CALL PTWO( BPTIME, RCARYT(PTR+2), POSITN,
     1                       FLOAT( RCARYP(PTR+2) ), PTYPE )
```

```
                        CALL PTWO( RCARYT(PTR+2), RCARYT(PTR+3),
     1                       FLOAT( RCARYP(PTR+2) ),
     2                       FLOAT( RCARYP(PTR+3) ), PTYPE )

                    ELSE

C                       Cut during the place
                        CALL PTWO( BPTIME, RCARYT(PTR+3),
     1                       FLOAT( RCARYP(PTR+2) ),
     2                       FLOAT( RCARYP(PTR+3) ), PTYPE )

                    ENDIF

                ENDIF

C                   Increment the pointer
                PTR = PTR + 4

C                   Check for the end of the input data
                IF ( PTR .GE. NRCARY ) RETURN

            ENDIF

        ELSE

C               Increment the pointer
            PTR = PTR + 4

C               Check for the end of the input data
            IF ( PTR .GE. NRCARY ) RETURN
            GOTO 10

        ENDIF

C
C           The main program loop
C

C           Check for the end of the plotting time
 20     IF ( RCARYT(PTR+3) .LE. EPTIME ) THEN

C               The period is fully plotable
            CALL PFOUR( RCARYT(PTR), RCARYT(PTR+1), RCARYT(PTR+2),
     1           RCARYT(PTR+3), FLOAT( RCARYP(PTR) ),
     2           FLOAT( RCARYP(PTR+1) ), FLOAT( RCARYP(PTR+2) ),
     3           FLOAT( RCARYP(PTR+3) ), PTYPE )

            PTR = PTR + 4

C               Check for the end of the input data
            IF ( PTR .GE. NRCARY ) RETURN
```

```
            GOTO 20

         ENDIF

C
C           Check to see how much of the final period is plotable
C
         IF ( RCARYT(PTR) .LT. EPTIME ) THEN

C              There is a partial plot
            IF ( RCARYT(PTR+2) .LE. EPTIME ) THEN

C                 The place is cut
               CALL PFOUR( RCARYT(PTR), RCARYT(PTR+1), RCARYT(PTR+2),
     1              EPTIME, FLOAT( RCARYP(PTR) ), FLOAT(RCARYP(PTR+1)),
     2              FLOAT(RCARYP(PTR+2)), FLOAT(RCARYP(PTR+3)), PTYPE )

            ELSE

               IF ( RCARYT(PTR+1) .LT. EPTIME ) THEN

C                    The move is cut, interpolate the cutoff
C                    position
                  POSITN = (EPTIME - RCARYT(PTR+1))*(RCARYP(PTR+2) -
     1                 RCARYP(PTR+1))/(RCARYT(PTR+2) - RCARYT(PTR+1))
     2                   + RCARYP(PTR+1)
                  CALL PTWO( RCARYT(PTR), RCARYT(PTR+1),
     1                 FLOAT( RCARYP(PTR) ), FLOAT( RCARYP(PTR+1) ),
     2                 PTYPE )
                  CALL PTWO( RCARYT(PTR+1), EPTIME,
     1                 FLOAT( RCARYP(PTR+1) ), POSITN, PTYPE )

               ELSE

C                    The pick is cut
                  CALL PTWO( RCARYT(PTR), EPTIME, FLOAT(RCARYP(PTR)),
     1                 FLOAT( RCARYP(PTR+1) ), PTYPE )

               ENDIF

            ENDIF

         ENDIF

         RETURN

         END
```

E.7 MODELCOM.FOR

```
      COMMON /MODEL/ MUPA, NUPA, MCPA, NCPA, MUPB, NUPB, MCPB, NCPB,
     1     MUPC, NUPC, MCPC, NCPC, MUPD, NUPD, MCPD, NCPD, RIDLET,
     2     RIDLEP, NRIDLE, RCARYT, RCARYP, NRCARY, RMOVET, RMOVEP,
     3     NRMOVE, BRTIME, ERTIME, CONVYN, CONVYT, NCONVY

      REAL MUPA(500), MCPA(500), MUPB(500), MCPB(500), MUPC(500),
     1     MCPC(500), MUPD(500), MCPD(500), RIDLET(500),
     2     RCARYT(500), RMOVET(500), BRTIME, ERTIME, CONVYT(500)

      INTEGER*2 NUPA, NCPA, NUPB, NCPB, NUPC, NCPC, NUPD, NCPD,
     1     NRIDLE, NRCARY, NRMOVE, RIDLEP(500), RCARYP(500),
     2     RMOVEP(500), NCONVY, CONVYN(500)
```

E.8 PLOTCOM.FOR

```
      COMMON /PLOT/ BPTIME, EPTIME, TYPE, TMPSCR, YLABEL

      CHARACTER*1 YLABEL

      REAL BPTIME, EPTIME, TMPSCR

      INTEGER*2 TYPE
```

Appendix F

DETAILED MODEL LISTING

F.1 FMS.DES

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                            ;
;                                                                            ;
;                                 FMS.DES                                    ;
;                                                                            ;
;                                    by                                      ;
;                                                                            ;
;                                Tim Martin                                  ;
;                                                                            ;
;                   5/ 7/85                         Ver 1.12                 ;
;                                                                            ;
;                                                                            ;
;       This is a description of a model to simulate the Virginia Tech      ;
;       Flexible manufacturing system.  The model is in the files           ;
;       FMS.MOD and FMS.EXP, and the FORTRAN subroutines are in             ;
;       the files PRIME.FOR, EVENT.FOR and TRACE.FOR.                       ;
;                                                                            ;
;                                                                            ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

This model simulates the Virginia Tech Miniature Flexible Manufacturing System (FMS). The FMS consists of four machining stations, an input conveyor and an exit conveyor. All parts are mounted on a pallet before they enter the system. Pallets are moved around the system by a robot in the center of the system. The machines and conveyors are equally spaced around the robot. The layout is shown below. The robot has a capacity of one pallet. All machines

```
                Machine D          Exit Conveyor
                        \          /
                         \        /
          Machine C ------ Robot ------ Input Conveyor
                         /        \
                        /          \
                Machine B          Machine A
```

are different and can not be substituted for any other machine. A pallet can only move to a machine that is in increasing alphabetical order from the machine the pallet is currently at. This is to prevent a deadlock from occurring. If, for example, the pallet at machine A needs to go to machine B and the pallet at machine B needs to go to machine A a lock up will occur and the system will

stall. The machine sequence for any pallet type is fixed. The processing time for a pallet type on a particular machine is fixed. After processing is done on a machine, the machine is brought to its home position. This time may be considered as an extra random time or it may be considered as part of the fixed processing time. (The time should be almost the same from pallet to pallet.) There may be any number of pallet types. Specifics about the model are explained below.

The model is written in the simulation language SIMAN. The model consists of three parts: the SIMAN model frame, the SIMAN experimental frame and FORTRAN subprograms. The model frame, stored in FMS.MOD, contains the model. The experimental frame, stored in FMS.EXP, contains the 'experiment' - or data. The FORTRAN subprograms, stored in PRIME.FOR, TRACE.FOR and EVENT.FOR, are used to input parameters at the beginning of a run, to generate a specialized trace of the runs and to perform some of the more difficult tasks of the model. The model has SIMAN statements that describe the hardware and software in the system and the relationships between the hardware and software. The experiment has SIMAN statements that control the simulation and has statements that define the variables used in the model. The control statements tell SIMAN how long the simulation is to last, the maximum values for many variables, whether or not to trace part of the simulation and which variables should be used for statistical output. The FORTRAN initialization and trace subprograms are mostly just I/O statements. The other FORTRAN subprograms (called events by SIMAN) take input parameters from the model frame and call SIMAN subroutines to obtain the desired change in the system. The FORTRAN subprograms must be linked to the SIMAN processor before they can be used.

The numbering system for the machines in the model is as follows:

Machine 1 is the robot.
Machines 2 through 5 are machine stations A through D.
The input and exit conveyors do not have a machine designation.

With two exceptions, this numbering is used for all of the parts of the model that are machine dependent. Often the machine number is added to a base to access the appropriate variable. An example is the queues 36 through 40. Queue 36 is used for the robot mini host and queues 37 through 40 are used for machine A through D's mini host, respectively. The values x1 through x5 are also used in addition to x6 through y0. (Where y = x + 1). The robot mini host is not used in the model at this time, so the first value of most values is not used. The storage is reserved for the robot mini host and should not be used for any other purpose. The exceptions to this numbering scheme are the numbering of the MACHINE resources and the pallet stations. Both of these use the same numbering scheme. The MACHINE resources' and the pallet stations' machines A through D are numbered consistently with the rest of the model. The input conveyor is defined as MACHINE(1) and station 1. The exit conveyor is defined as MACHINE(6) and station 6. The robot does not have a station number or a MACHINE resource (it uses the transporter ROBOT).

The model consists of 6 pallet stations and 2 stations for entity routing:

Station 1 is the input conveyor,
Stations 2-5 are the machine stations,
Station 6 is the exit conveyor,
Station 7 is the entry point for the mini host polling entities at the start of a run, and

To move a pallet from one pallet station to another, the robot is requested at the current station. When the robot arrives, a pick is performed. A pick tells the robot to pick up the pallet. The pick is modeled simply as a time delay. The robot is then moved to the desired station. After arriving at the station, a place is performed. A place tells the robot to place the pallet on the machine or conveyor. The place is modeled by a time delay. The robot is then released so other pallets can use the robot.

The attributes for all pallets are defined as:

A(1) is the time after the start of the simulation that the pallet entered the system.
A(2) is the type of pallet plus 50. The pallet type defines the sequence of machines and processing time for the pallet.
A(3) is the station the pallet is at.
A(4) is the processing time at station A(3) - This may or may not include the time it takes for the machine to go to its home position after processing is done. The time to get to its home position can be represented as a uniform distribution.
A(5) is the number of work stations the pallet has been to. The input and exit conveyors count for this value.
A(6) is used as an index when getting the next processing time. This attribute can be used for other temporary storage.
A(7) is the pallet number - the pallets are numbered in the order they entered the system, starting at one.
A(8) is the station that a pallet is moving to.
A(9) indicates whether the pallet is waiting to updated and moved (A(9) = 0), waiting to be moved (A(9) = .5) or is in process (A(9) = 1). When the pallet is updated, the next station and processing time is found for the pallet.
A(10) is the number of commands needed at station A(3).
A(11) is used by the message sending event to store which queue the message is to be placed in after it is sent.
A(12) is used by the message sending event to store which signal code is to be sent when the message is received.
A(13) is used by the message sending event to store which communication line is being used for the transmission
M is the station the pallet is at

The attributes for the system host polling entity are:

None are used.

The attributes for the system host interrupt entities are:

A(1) is used to hold the number of commands that can be output the remainder of this interrupt routine.
M is the machine number of the mini host the interrupt routine "talks" to.

The attributes for the mini host polling entities are:

A(1) is the number of commands input from the system host since the last XON sent by the mini host.
A(2) is used to indicate whether or not the last command has been received from the system host for the current pallet. If A(2) is 0 the last command has not yet been received. If A(2) is 1 the last command has been received (the end of the file was reached).
A(3), A(4) are currently unused.
A(5) is used to hold the machine number of the mini host. This value is used to set the M attribute after the entity has moved through a STATION block. (Moving through a STATION block sets the value of M to the STATION number.)
A(6) is used for scratch storage. It is used to get the parameter set the range of values for the time it takes to bring a machine to its home position is stored in. It can be used for other temporary storage.
M is the machine number of the mini host

The attributes for the axis controller entities are:

M is the machine number of the mini host

The resources used are:

1-6 are MACHINE(1-6). CAUTION! Read the note below! These are used to control the availability of all of the stations. If the resource is unavailable, the machine is busy with a part, has a part coming to it or has a part waiting to be picked up. If the resource is available, then the pallet can seize the resource and request the robot to move it to the station. The output conveyor is always available when the robot is free.
NOTE!: MACHINE(1) corresponds to the input conveyor and not the robot. The robot does not have a "MACHINE" resource! This is different from all other uses of the numbering 1-6! Machine 1 is usually considered to be the robot. Machines 2 through 5 always mean machine stations A through D. The input and exit conveyors

usually do not have a machine designation.

7 SYSHOST: represents when the system host is busy. This resource is used to control the "sharing" of the system host being the polling routine and the interrupt routines. It is used to make sure the system host CPU is not being used by more the one routine at any one time.

8-12 MINIHOST: These resources represent when each minihost is busy. It is used to make sure each minihost is only doing one thing at a time.

13-17 AXISCONT: These resources represent when the axis controllers are busy. Currently, they are busy whenever a pallet is being worked on.

The global variables (X(i)'s) are defined as:

X(1-5) are used to indicate that machines 1-5 are busy. If X(i) is 1, the machine is busy with a part. These variables are used only to keep statistics on machine utilization. X(1) is used for the robot.

X(6) is used to keep track of the number of pallets on the input conveyor. The number of pallets on the input conveyor is equal to the sum of the number of pallets in QUEUE(1) and the number of busy units of MACHINE(1). MACHINE(1) is the input conveyor, and QUEUE(1) is the queue for pallets waiting for the input conveyor.

X(7) is used to hold the station the robot is at, or the last station the robot was at. This variable is used only for data collection for plot outputs.

X(8) is used as a pointer to indicate where in the task list to start checking for an idle pallet. The first time the task list is searched, X(8) is 1. If a pallet is found that is idle, but can not be moved, X(8) is set to the task list position after the position of the 'stuck' pallet and the task list is searched from that point on for another idle pallet.

X(9) is used to temporarily hold the pallet number of a pallet that is having its entry on the task list updated.

X(10) is used to store the utility of the input conveyor - system host serial line

X(11-15) are used to store the utility of the serial line into the mini hosts

X(16-20) are used to store the utility of the serial line out of the mini hosts

X(21-25) contains the number of commands remaining to be output to a mini host from the system host. X(21) is used for the robot mini host and X(22) through X(25) are used for the machine A through machine D mini host, respectively.

X(26)-X(30) are currently unused.

X(31-35) contains the average machine processing time for each command.

X(31) is used for the robot and X(32) through X(35) are used for machine A through machine D, respectively.

X(36-50) are used for passing parameters to and from event routines. At this time these are only used to send and receive messages. These variables can also be used for temporary scratch storage.

The queues used in the model are:

1-6 Wait for station q to become available. The numbering is the same as used for the station numbers.

7 Wait for the robot to pick up a pallet.

8 The task list. This is where the status of all of the active pallets in the system is stored. Copies of pallet entities are stored in this queue.

9 The input queue. After a pallet physically moves to the front of the input conveyor its presence is noted in the input queue until there is room on the task list for the pallet. This is a queue that is part of the software of the system host.

10 The system host 'waits' here while a pallet is being processed. Pallet processing is done only on idle pallets on the task list. Pallet processing is updating the pallet's entry on the task list or checking to see if the pallet can move on, and if so, moving the pallet to the next station.

11 The system host 'waits' here until a signal telling the system host that one of the things it polls for has happened. This is used by the model so the system host does not need to continuously poll in the model. This speeds up model run times considerably since the system host spends most of its time polling. The signal is sent when a pallet moves to the front of the input conveyor or when a pallet finishes processing at a machine station. An entity representing the system host's polling functions is placed here at the beginning of the run.

12 Wait for the system host to become available for polling operations and check from the beginning of the task list.

13 Wait for the system host to become available for polling operations and check from the last checked position on the task list.

14 Wait to get the system host before bringing the robot to the current station. The robot will be moving without a pallet.

15 Wait to get the system host when moving the robot from station to station. The robot will be moving with a pallet.

16 Wait to get the system host for loading the data buffer from disk.

17 The system host interrupt routine entity waits here while waiting for an ACK from the mini host after the mini host has received data from the system host and the system host has more data to send.

18 Same as queue 17, except the system host has no more data to send.

19 Wait to get the system host to check to see if another message was received from the mini host.

20 The system host interrupt entity waits here for input from the

mini host. It waits here after the system host interrupts are enabled. The system host interrupt routine entities are placed here at the beginning of a run.

21-25 Messages from the system host to the mini hosts are stored here. Queue 21 is used for messages to the robot mini host and queues 22-25 are used for messages to the machine A-D mini hosts, respectively.

26 Wait for the system host to become available for processing in the interrupt service routine.

27 The system host interrupt entities wait here until the system host enables interrupts from the mini hosts.

28 The mini host polling entity waits here for input from the system host.

29 The mini host polling entity waits here until the processing is done on a pallet and after the mini host has output all of the commands for the pallet to the axis controller.

30 currently unused.

31-35 Messages from the mini hosts to the system host are stored in these queues. Queue 31 is used for messages from the robot mini host and queues 32-35 are used for messages from the machine A-D mini hosts, respectively.

36-40 The minihost polling entities wait here to get the mini host to process input from the system host.

41-45 Pallet entities wait here until the process on the machine they are on is finished. Queue 41 is for the robot and queues 42-45 are for the machine stations. The robot mini host is currently not used to help move pallets. All of the storage is reserved for later use by the robot min host.

51 The axis controller entities wait here for input from the mini hosts. The axis controller entities start the run here.

52-55 are currently unused.

56-60 The axis controller polling entities wait here to get the axis controller to process input from the system host.

61-65 The mini host polling entities wait here to get the mini host after the process on the machine is done.

The signal codes used by the model are:

1 is used when one of the conditions that the system host polls for changes. This will 'wake up' the system host if it is waiting for something to happen. The method used to poll is as follows: An entity representing the polling CPU is put in a queue waiting for a status change. When the status changes the event that causes the status to change sends the signal code the CPU entity is waiting for. The CPU then goes through its normal polling procedure. When it is done with that it polls once more to check if anything has changed since its last check. If nothing has changed then it returns to the queue to wait for another signal. The system host polls the following activities: the arrival of a pallet to the beginning of the input queue and if a pallet is ready to move to

another station.

2-10 are currently unused.

11-15 are used to indicate that input from the system host has been received by the mini host. Signal code 11 is used to signal the robot mini host and signal codes 12-15 are used to signal the machine A-D mini hosts, respectively.

16-20 are currently unused.

21-25 are used to indicate that input from the mini host has been received by the system host.

26-30 are currently unused.

31-35 are used to indicate that the interrupts have been enabled on the system host to allow input from the mini host.

41-45 are used to indicate that a process is done.

46-50 are used to indicate that data has been sent from the mini host has been received by the axis controllers.

51-55 are used by the axis controllers to tell the mini host that it is done with the current pallet.

The parameter sets are defined as:

1 is the pallet arrival time mean (1,1).

2 is the cumulative probability that a pallet is of a certain type. The odd values in this set are the cumulative probabilities of the pallet types in the even values. There must be an entry in this set for all possible types of pallets.

3 contains the trace status. If (3,1) is 1, the trace is on. If (3,1) is 0, the trace is off.
The rest of parameter set 3 is a sequence of times that represent the times when the trace status changes. There MUST be a change at or after the end of the run. (The program will crash if there is not a change after the end of the run).

4 is the pick and place times. (4,1) is the pick time and (4,2) is the place time.

5-8 are the minimum and maximum time it takes to get a machine to its home position. Parameter set 5 is used for machine A, 6 for B, 7 for C and 8 for D. (i,1) is the minimum, and (i,2) is the maximum. If both of these values are set to zero, then the time to bring the machine to its home position should be included in the processing time for each machine.

9 is used to hold the transmission delay times.
(9,1) is used to hold the time it takes to transmit one character to the system host from the input conveyor.
(9,2) is the time to transmit one character either way between the system host and any of the mini hosts.
(9,3) is the time to transmit one character either way between the mini host and the axis controller.

10 contains the processing delay times for the system host (including disk access time)
(10,1) is the time it takes the system host to check the task list
(10,2) is the time needed to transfer an entry in the input queue

to the task list.
(10,3) is the set up time for a robot move.
(10,4) is the time needed to read one command from disk.
(10,5) is the time it takes to be interrupted and to read one character in the interrupt routine.

11 contains the processing delays for the mini hosts.
(11,1) is percent of the CPU time taken by the timer interrupt.

12 contains the processing delays for the axis controllers. currently there are none.

13 is the range of times until an axis will be ready to receive data from the mini host. The axis will only receive data between processing of commands.

14-50 are currently unused.

51 up are used to define the machine sequences and processing times on each machine for each pallet type. The set used for each pallet type is the pallet type number plus 50.
(i,1-5) are used for the part sequence. All parts start at station 1, so this is assumed and must not be included. Each part must end at station 6, and this must be included in the part sequence.
(i,6-10) are used for the processing time on the station stored in j-5. The processing time may or may not include the time it takes for the machine to go to its home position after processing is done. See parameter set 3 for details.
(i,11-15) are used for the number of commands used to process the pallet on each machine. A number of commands must be supplied for the exit conveyor, it does not matter what it is.

The random number streams used are:

1 is used for the time between pallet arrivals.

2 is used for the time it takes a machine to get to its home position.

3 is used to determine the part type.

4 is used in determining the time it will take before an axis controller is ready to receive data from the mini host.

Counter number 1 is used to count the number of pallets that have entered the system. This is used to mark the pallet number (A(7)).

The FORTRAN subroutines fall into several categories. There are initialization and finalization routines and events. The events can be broken up into two groups: trace events and new command events. All of the FORTRAN subroutines used are (functionally) described below.

The initialization routine is called PRIME. PRIME has two functions: to input any changes in the parameter set that the user wants and to initialize the trace. The subroutine to input any changes the user wants in the parameter set

is called ASKP. ASKP only works on values in the parameter sets, other system variables can not be changed. Only values defined in the experimental frame can be changed, no new values can be added to the parameter set. The change remains in effect until the parameter is changed again or until the SIMAN program is exited. The trace initialization routine is called SETTR. SETTR creates an entity that will be used to turn the trace on and off and SETTR schedules this entity to arrive at the trace status change event at the first trace change time.

The finalization routine is called WRAPUP. It only prints a form feed to the output to make sure that the SIMAN summary report is on a different page than the parameter inputs or any trace output.

Calls to FORTRAN subroutines are made using the SIMAN command EVENT: i, where i is a number that is used to determine which subroutine is to be called. The subroutine EVENT maps the SIMAN event numbers to FORTRAN subroutines calls.

The trace is generated by calls to FORTRAN subroutines that print out the status of the system at the time of the call. Different trace events are put at different points of the model so the flow of pallets through the system can be traced. The specifics as to what each trace routine prints are listed below.

The trace is turned on and off using a FORTRAN event (Event number 11). An entity is created in PRIME that is scheduled to arrive at event 11 the next time a trace status change is to occur. Whenever the entity arrives at the event it changes the trace status and gets a new trace change time from parameter set 3. The entity's attribute 1 is used to point to the next trace status change in parameter set 3. Parameter set 3 must have a trace change event scheduled after the end of the run or the run will bomb at the last trace change time.

The new command events are all events that are called from the model when the desired function is needed. These events are like extensions to the SIMAN language. They perform tasks that would be difficult or impossible to implement in the SIMAN model frame. They also increase the readability of the model since complicated procedures that don't aid in the understanding of the model are moved to the FORTRAN routines. The added events are listed below.

The FORTRAN events are:

Events 1-5 are trace events used by the HL model.

1 ARRIVAL: called whenever a pallet arrives at the system. Prints the pallet type and machine sequence of the arriving pallet and prints the number of pallets waiting on the input conveyor.

2 MOVE: called when a pallet is ready to move to another station. Prints that the pallet is ready to move and prints which machines are unavailable.

3 ROBOT: called when a pallet gets use of the robot. Prints that the pallet has been picked up by the robot.

4 DWROBOT: called when the pallet is done with the robot and is ready to start processing on a machine. Prints that the pallet has arrived at the machine, prints the pallet's processing time on the

machine, prints which pallets are waiting for a machine or the robot and prints which machines are unavailable.

5 FINISH: called when a pallet is leaving the system. Prints the pallet's time in the system, prints which pallets are waiting for a machine or the robot and prints which machines are unavailable.

Events 5-10 are trace events used by the FMS model.

6 FARRIV: called whenever a pallet arrives at the system. Prints the pallet type and machine sequence of the arriving pallet and prints the number of pallets waiting on the input conveyor.

7 FMOVE: Called when a pallet is ready to move to another station. Prints that the pallet is ready to move and prints which machines are unavailable.

8 FDWROB: called when the pallet is done with the robot and is ready to start processing on a machine. Prints that the pallet has arrived at the machine, prints the pallet's processing time on the machine, prints the task list and prints which machines are unavailable.

9 FFINIS: Called when a pallet is ready to leave the system. Prints the pallet's time in the system, prints the task list and prints which machines are unavailable.

10 FROBOT: called when a pallet gets use of the robot. Prints that the pallet has been picked up by the robot.

The rest of the events are used only in the FMS model and are new command events. (Event 11 is also used by HL).

11 CHGTR: used by both HL and FMS to change the trace status.

36 SNDMES: used to transmit a message. It requires the following input:

- X(36) is the transmission time
- X(37) holds the queue the message is to be placed in
- X(38) is the signal code to be sent. If X(38) is 0 then no signal code is sent
- X(39) is the length of the message. The maximum is 10. The length of the message is not used in any way by this event to calculate the transmission time. If X(39) is 0 then the attributes of the invoking entity are used as the message.
- X(40)-X(49) is the message
- X(50) is the transmission line to use

37 RCVMES: used to read a message. The input required is that X(37) be set to the queue the message is to be read from.
The outputs are:

- The message is removed from the queue
- X(39) is set to the length of the message. If X(39) is 0 on return, then there was no message in the queue.
- X(40)-X(49) contains the message

50 UPTASK: updates the task list with the arriving entity. If the pallet is already on the task list its old value is over written by its current value.

51 COMERR: prints an error message when there is a communication

error when the mini host sends data to the system host.

98 SETENT: sets up the CPU entities. This event is called at the beginning of each FMS run to initialize the CPU entities. It is not in PRIME because it would cause HL to crash when run. The call to the event is made by an ARRIVAL entity in the FMS experimental frame. The entities setup are listed below:

- The system host polling entity is placed in queue 11. None of its attributes are set.
- The system host interrupt routine entities are placed in queue 20. Each entity's M attribute is set to the interrupt service routine it corresponds to.
- The mini host polling entities enter the model at station 7. Each entity's M attribute and A(5) attribute are set to the machine number the entity corresponds to.
- The mini host timer interrupt entities enter the model at station 8. Each entity's M attribute and A(5) attribute are set to the machine number the entity corresponds to.
- The axis controller polling entities are placed in queue 51. Each entity's M attribute is set to the machine number it represents.

99 MESARV: processes a message that has arrived at a component. This event is scheduled from event 36 (Send a message). Sends the signal code and places the message in the appropriate queue.

F.2 FMS.MOD

```
BEGIN;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                     ;
;                                                                     ;
;                               FMS.MOD                               ;
;                                                                     ;
;                                 by                                  ;
;                                                                     ;
;                              Tim Martin                             ;
;                                                                     ;
;                    5/ 7/85                 Ver 1.23                 ;
;                                                                     ;
;                                                                     ;
;              Model to simulate the Virginia Tech FMS.               ;
;                      See FMS.DES for details.                       ;
;                                                                     ;
;                                                                     ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                     ;
;                                                                     ;
;         Pallet arrival block - Controlled by the pallet entity      ;
;                                                                     ;
;                                                                     ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
        CREATE: EX (1,1):-      Pallets arrive based on an exponential
                        -       distribution (The mean is stored in P(1,1) )
        MARK( 1 );              Store the arrival time in A(1)
        COUNT: 1;               Count the number of pallets that have arrived
        ASSIGN: A(7) = NC(1);   Mark this as the nth pallet
        ASSIGN: M = 1;          The input conveyor is station 1
        ASSIGN: A(3) = 1;       Set A(3) to the current station
        ASSIGN: A(2) = DP(2,3) + 50;  Find out which pallet type it is. The
                               ;types are defined by the discrete
                               ;distribution table found in parameter set 2
        ASSIGN: X(6) = NQ(1) + NR(1) + 1;  Compute the number of pallets on the
                               ;input conveyor (include the arriving pallet)
        EVENT: 6;               Trace event 6 - pallet arrival

        QUEUE, 1;               Wait on the input conveyor until the
        SEIZE: MACHINE(1);          end of the conveyor is reached

        EVENT: 7;               The pallet is ready to move to the next
               ;                    station
```

```
        ASSIGN: X(36) = P(9,1);  Prepare to send a message: The transmission
                              ;       time is found in P(9,1)
        ASSIGN: X(37) = 9;       Put the message in the input queue (Queue 9)
        ASSIGN: X(38) = 1;       Send signal code one after the message is sent
                        ;            (Alert the polling system host of the
                        ;             message)
        ASSIGN: X(39) = 0;       Use the attributes of the incoming pallet as
                        ;            the message
        ASSIGN: X(50) = 10;      Use the serial line between the input conveyor
                         ;           and the system host
        EVENT: 36:-              Send the message
        DISPOSE;                 Don't do anything else
;
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                            ;
;                                                                            ;
;            The Task List - Stores pallet entities                          ;
;                                                                            ;
;                                                                            ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;
TASKLIST QUEUE, 8: DETACH;       Add the pallet to the task list
;
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                            ;
;                                                                            ;
;            The system host model - controlled by the system host           ;
;                 polling entity                                             ;
;                                                                            ;
;                                                                            ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;


;
;            CHecK to see if there is RooM on the TaSK list
;
CHKRMTSK BRANCH, 1:-             See if there is room on the task list for
                   -                  another pallet and if there is a pallet
                   -                  waiting to get on the task list
            IF, ( ( NQ(8) .LT. 4 ) .AND. ( NQ(9) .GT. 0 ) ), PUTTSKLS:
                                -There is room on the task list and there is a
                                -     pallet waiting to get on the task list
            ELSE, CHKTSKLS;      There is no more room on the task list or no
                          ;           pallet is waiting


;
;            PUT the pallet that is waiting in the input queue on the TaSK LiSt
```

```
;
PUTTSKLS REMOVE: 1, 9, TASKLIST;  Take the top pallet off the input queue and
                              ;       place it on the task list
         DELAY: P(10,2):¬         Delay by the time it takes to put the pallet
                        ¬              on the task list
         NEXT( CHKRMTSK );        Then go and CHecK to see if there is more RooM
                         ;             on the TaSK list

;
;             CHeck the TaSK LiSt
;
CHKTSKLS ASSIGN: X(8) = 1;        Set where to start the search
         DELAY: NQ(8)*P(10,1);    Delay for the time it takes to check the task
                              ;         list
;
;             CHeck the TaSk List Again - reentry point after a pallet that
;                  can't be moved is found
;
CHKTSKLA SEARCH, 8, X(8):¬        Search the task list starting at element X(8),
              ( A(9) .LT. 1 );         looking for a pallet that is idle
         BRANCH, 1:¬              Check to see if an idle pallet was found
              IF, ( J .EQ. 0 ), SYSHODUN:¬  None was found
              ELSE, GETPAL;       A pallet was found

;
;             GET the idle PALlet
;
GETPAL   REMOVE: J, 8, PROCPAL;   Get the pallet from the task list and send it
                              ;         to be processed
         ASSIGN: X(8) = J + 1;    Set the next starting point into the task list
                              ;         to just after where the last pallet
                              ;         checked is
         RELEASE: SYSHOST;        Free the system host
         QUEUE, 10: DETACH;       Wait until the pallet has been processed

;
;             CHeck ANTheR pallet because the last pallet checked could not be
;                  moved
;
CHKANOTR QUEUE, 13;
         SEIZE: SYSHOST;          Get the system host
         BRANCH, 1:¬              Then go and check the task list again if:
              IF, ( X(8) .LE. NQ(8) ), CHKTSKLA:¬there are more pallets to check
              ELSE, SYSHODUN;                    otherwise, the SYStem HOst is
                            ;                         DONe

;
;             The SYStem HOst is DONe
;
SYSHODUN RELEASE: SYSHOST;        Free it up
```

```
        QUEUE, 11;
        WAIT: 1;                 Wait for something to signal the polling
            ;                         system host
;
;           GET the polling SYStem HOst
;
GETSYSHO QUEUE, 12;
        SEIZE: SYSHOST:-         Get the system host
                    -
        NEXT( CHKRMTSK );        Run the system host polling program

;
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                          ;
;                                                                          ;
;           The system host model - the part 'controlled' by the pallet    ;
;                                                                          ;
;                                                                          ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;


;
;           PROCess an idle PALlet on the task list
;
PROCPAL  BRANCH, 1:-             Check to see if the current pallet needs to be
                -                     updated to where it is going next and to
                -                     see if it can move to its next station
            IF, ( A(9) .EQ. 0 ), UPDATE:- The pallet needs to be updated
            IF, ( NR(M) .LT. MR(M) ), SETPALBZ:
                                -The pallet is updated and can move on
            ELSE, NOTTISPL;      The pallet is stuck


;
;           UPDATE the pallet's status
;
UPDATE   ASSIGN: A(5) = A(5) + 1;          Increment the sequence number
         ASSIGN: A(8) = P( A(2), A(5) );   Get the next station number
         ASSIGN: A(6) = A(5) + 5;          Index the processing time
         ASSIGN: A(4) = P( A(2), A(6) );   Get the next processing time
         ASSIGN: A(6) = A(5) + 10;         Index the number of commands
         ASSIGN: A(10) = P( A(2), A(6) );  Get the number of commands
         ASSIGN: M = A(8);       The next machine must be a function of M
         ASSIGN: A(9) = .5:-     The next station has been specified

         NEXT( PROCPAL );        PROCess the PALlet again


;
;           NOT ThIS PaLlet - the pallet being checked can not move to its
;                next station
```

```
;
NOTTISPL REMOVE: 1, 10, CHKANOTR:-This pallet is stuck, check the task list for
                                 -     another pallet
        NEXT( TASKLIST );        Put the current pallet back on the task list

;
;            SET the current PALlet BuSy
;
SETPALBZ ASSIGN: A(9) = 1;       A(9) indicates that this pallet is busy
        EVENT: 50;               Update the task list

        QUEUE, M;                GET the next STAtion and the RoBot
        SEIZE: MACHINE( M );     Get the next machine
        ASSIGN: M = A(3);        Restore the current station number

        QUEUE, 14;               Get the system host
        SEIZE: SYSHOST;
        DELAY: P(10,3);          The time it takes to calculate a robot move
        RELEASE: SYSHOST;        Let the system host do other things

        QUEUE, 7;                Wait for the robot
        REQUEST: ROBOT;
        ASSIGN: X(7) = M;        The robot is at station M
        EVENT: 10;               Trace event 10 - a pallet is picked up by
                ;                the robot

        DELAY: P(4,1);           Perform a pick
        ASSIGN: X(7) = M+.5;     The robot is leaving station M

        RELEASE: MACHINE(M);     The old machine is now available
        ASSIGN: X(6) = NQ(1) + NR(1); Compute the number of pallets on the
                                    ; input conveyor

        QUEUE, 15;               Get the system host
        SEIZE: SYSHOST;
        DELAY: P(10,3);          The time it takes to calculate a robot move
        RELEASE: SYSHOST;        Let the system host do other things

        TRANSPORT: ROBOT, A(8);  Move the pallet to its next station

;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                            ;
;                                                                            ;
;            The machine stations - controlled by the pallet entity          ;
;                                                                            ;
;                                                                            ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
        STATION, 2-5;
        ASSIGN: X(7) = M;        The robot is at the current station
```

```
        ASSIGN: A(3) = M;        Set A(3) to the current station
        EVENT: 50;               Update the task list

        DELAY: P(4,2);           Perform a place
        ASSIGN: X(7) = M+.25;    The robot is done at station M

        FREE: ROBOT;             Done with the robot for now
        EVENT: 8;                Trace event 8 - arrival at a machine station

        QUEUE, 16;               Wait to get the system host
        SEIZE: SYSHOST;
        DELAY: A(10) * P(10,4);  Read the commands from disk
        ASSIGN: X(M+20) = A(10)+1;  Set the buffer length to one plus the
                                 ;    number of commands to be sent. The extra
                                 ;    'command' is the end of file marker.
        ASSIGN: X(M+30) = A(4) / A(10);    Calculate the average time it takes
                                      -    to process a command
        SIGNAL: M + 30;          Enable serial receive interrupts from the mini
                      ;               host
        RELEASE: SYSHOST;        Done with the system host for now
        REMOVE: 1, 10, GETSYSHO; GET the SYStem HOst to check the task list

        QUEUE, M + 40;           Wait for the process to finish
        WAIT: M + 40;

        ASSIGN: A(9) = 0;        Set the pallet to idle
        SIGNAL: 1;               Wake up the polling system host

        EVENT: 7;                Trace event 7 - a pallet is done at a machine
               ;                      and is ready to move to another station
        EVENT: 50:-              Update the task list
        DISPOSE;                 Only need the value on the task list
;
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                              ;
;                                                                              ;
;          Station 6 is the exit conveyor - controlled by the pallet          ;
;               entity                                                         ;
;                                                                              ;
;                                                                              ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
        STATION, 6;
        ASSIGN: X(7) = 6;        The robot is at the exit conveyor

        RELEASE: MACHINE( 6 );   The output conveyor is not committed

        DELAY: P(4,2);           Perform a place
        ASSIGN: X(7) = 6.25;     The robot is done at the exit conveyor
```

```
        FREE: ROBOT;               Done with the robot
        REMOVE: 1, 10, GETSYSHO; GET the SYStem HOst to check the task list

        ASSIGN: X(9) = A(7);       Set up to remove the pallet from the task list
        SEARCH, 8: ( A(7) .EQ. X(9) );Find where on the task list the pallet is
        REMOVE: J, 8, LEAVEFMS:- Send the old task list value to tell the
                               -      model the pallet is done
        DISPOSE;                   The pallet is done

LEAVEFMS TALLY:1, INT(1);         Get time in the system
        EVENT: 9:-                 Trace event 9 - A pallet is leaving the system
        DISPOSE;                   The present pallet is done

;
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                                ;
;                                                                                ;
;          The System Host Interrupt Service Routine - controlled               ;
;               by the system host interrupt entities                            ;
;                                                                                ;
;                                                                                ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;


;
;          WAIT FOR INPut from the mini host
;
WAIT4INP RELEASE: SYSHOST;        Let the system host do other things until an
                     ;                  input arrives

        QUEUE, 20;                 Wait for input
        WAIT: M+20;

        QUEUE, 26;                 Get the system host
        PREEMPT: SYSHOST;

        DELAY: P(10,5);            Interrupt time and the time it takes to read
                   ;                    the input character

        ASSIGN: X(37) = M+30;      Set up to read the input character
        EVENT: 37;                 Read the character

;
;          READ and process the INPut character
;
READINP  BRANCH, 1:-               Check to see what the input character was
              IF, ( X(40) .EQ. 19 ), WAIT2PRO:-  XOFF was received
              IF, ( X(40) .EQ. 17 ), XON:-       XON  was received
              IF, ( X(40) .EQ. 15 ), PROCDONE:-  SO   was received
```

```
               ELSE, ERROR;                An illegal character was received

;
;              XON - The system host has received an XON character from the
;                    mini host.  Output the commands requested.
;
XON       ASSIGN: A(1) = 6;          The maximum number of commands that can be
                        ;                sent at one time
          BRANCH, 1:¬                Check to see if there are that many commands
                    ¬                     left to be sent
               IF, ( X(M+20) .LT. A(1) ), NOTENDAT:¬   There is not enough data
                                        ¬to send all of the possible commands
               ELSE, CHKCSENT;       There is enough data

;
;              NOT ENough DATa
;
NOTENDAT ASSIGN: A(1) = X(M+20);  Set the number of commands to be sent

;
;              CHecK to see if a Command can be SENT
;
CHKCSENT BRANCH, 1:¬               See if the end of file is reached; and if not,
                    ¬                     if the total number of commands that can
                    ¬                     be sent now have been sent
               IF, ( ( X(M+20) .EQ. 1 ) .AND. ( A(1) .NE. 0 ) ), DONETRAN:
                                       ¬The end of file indicator is to be sent next
                                       ¬     and another command can be sent this
                                       ¬     time, go to DONE TRANsmitting
               IF, ( A(1) .EQ. 0 ), WAIT2PRO:¬    If the maximum number of
                                       ¬     commands have been output, wait for
                                       ¬     another XON
               ELSE, SENDMORE;       Another command can be sent to the minihost

;
;              SEND MORE data to the minihost
;
SENDMORE ASSIGN: X(36) = 27*P(9,2);    Set the transmission time (27 characters
                                  ;         are to be sent)
         ASSIGN: X(37) = M+20;     Place the message in this queue
         ASSIGN: X(38) = M+10;     Set the signal code to be sent
         ASSIGN: X(39) = 1;        Length of the message to be sent
         ASSIGN: X(40) = X(M+30); Processing time for the command being sent
                               ;       (This is the message sent)
         ASSIGN: X(50) = M+10;     Use the serial line into the mini host
         EVENT: 36;                Send the message

         QUEUE, 17;                Wait to receive the ACKnowledgement
         WAIT: M+20;

         ASSIGN: X(37) = M+30;     Read the message sent
```

```
          EVENT: 37;               Assume the message was an ACK

          ASSIGN: X(M+20) = X(M+20) - 1;     Decrement the total number of
                                    ;          commands to be sent
          ASSIGN: A(1) = A(1) - 1:¬     Decrement the number of commands to be
                                  ¬          sent this interrupt routine
          NEXT( CHKCSENT );        See if more commands are to be sent

;
;              DONE TRANsmitting - The end of the command file has been reached,
;                   send the end of file 'command' and quit
;
DONETRAN ASSIGN: X(36) = 3*P(9,2);     Send 3 characters: EOF, CR, checksum
          ASSIGN: X(37) = M+20;     Where the message is being sent
          ASSIGN: X(38) = M+10;     Signal code to be sent
          ASSIGN: X(39) = 1;        Length of the message
          ASSIGN: X(40) = 0;        No more data
          ASSIGN: X(50) = M+10;     Use the serial line into the mini host
          EVENT: 36;                Send the message

          QUEUE, 18;                Wait for the ACKnowledgement
          WAIT: M+20;

          ASSIGN: X(37) = M+30;     Set up to read the ACK
          EVENT: 37;                Read the assumed ACK

;
;              WAIT TO PROcess another input
;
WAIT2PRO RELEASE: SYSHOST;         Done with the system host for now

;
;              CHecK to see if there is ANother INput
;
CHKANIN   QUEUE, 19;               Wait to get the system host
          PREEMPT: SYSHOST;

          ASSIGN: X(37) = M+30;    Prepare to attempt to read a message
          EVENT: 37;               Attempt to read a message

          BRANCH, 1:¬              See if a message has been sent
               IF, ( X(39) .EQ. 0 ), WAIT4INP:¬   No message was received
               ELSE, READINP;      A message was received

;
;              An ERROR has occurred - an illegal character was received
;                   from the mini host
;
ERROR     EVENT: 51;               Print an error message
```

```
;
;               The PROCess is DONE
;
PROCDONE SIGNAL: M+40;              Tell the waiting pallet that it is done
         RELEASE: SYSHOST;          We're done with the system host

         QUEUE, 27;                 Wait for the interrupts to be enabled
         WAIT: M+30:

         NEXT( CHKANIN );           Check to see if an input has been received

;
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                                ;
;                                                                                ;
;               The Mini Host Model - controlled by the mini host polling        ;
;                    entity                                                      ;
;                                                                                ;
;                                                                                ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;


;
;               SEND an XON character to the system host
;
SENDXON  ASSIGN: X(36) = P(9,2);    Set the transmission time
         ASSIGN: X(37) = M+30;      The queue the message goes to
         ASSIGN: X(38) = M+20;      The signal code to be sent
         ASSIGN: X(39) = 1;         Length of the message
         ASSIGN: X(40) = 17;        The message - an XON
         ASSIGN: X(50) = M+15;      Use the serial line from the mini host
         EVENT: 36;                 Send the XON

         DELAY: P(9,2);             Wait until the XON is completely sent

         ASSIGN: A(1) = 0;          Set the number of commands input so far
         ASSIGN: A(2) = 0;          The last command has not been received yet


;
;               INput MORE data from the system host
;
INMORE   RELEASE: MINIHOST(M);      The minihost is done for now

         QUEUE, 28;                 Wait for input from the system host
         WAIT: M+10;

         QUEUE, M+35;               Get the mini host
         SEIZE: MINIHOST(M);
```

```
         ASSIGN: A(1) = A(1)+1;   Count the number of commands input

;             Send an ACK to the system host
         ASSIGN: X(36) = P(9,2); Set the transmission time
         ASSIGN: X(37) = M+30;    The queue the message goes to
         ASSIGN: X(38) = M+20;    The signal code to be sent
         ASSIGN: X(39) = 1;       Length of the message
         ASSIGN: X(40) = 6;       The message - an ACK
         ASSIGN: X(50) = M+15;    Use the serial line from the mini host
         EVENT: 36;               Send the ACK

         DELAY: P(9,2);           Wait until the ACK is completely sent

         SEARCH, M+20, NQ, NQ:-   See if the last command was
              A(1) .EQ. 0;             the last command to be sent

         BRANCH, 1:-              Check to see if any more commands are coming
                   -                    now
              IF, J .NE. 0, EOF:- There are no more commands to be sent
              IF, A(1) .GE. 6, BUFFULL:-    The mini host buffer is full
              ELSE, INMORE;       There are more inputs coming now

;
;             The End Of the command File has been reached - all of the
;                  commands needed for this pallet have been received
;
EOF      REMOVE: J, M+20, DISPOSE; Remove the end of file marker from the
                            ;           command queue
         ASSIGN: A(2) = 1;        Mark that the end of the file has been reached

;
;             The BUFFer is FULL, send an XOFF to the system host
;
BUFFULL  ASSIGN: X(36) = P(9,2); Set the transmission time
         ASSIGN: X(37) = M+30;    The queue the message goes to
         ASSIGN: X(38) = M+20;    The signal code to be sent
         ASSIGN: X(39) = 1;       Length of the message
         ASSIGN: X(40) = 19;      The message - an XOFF
         ASSIGN: X(50) = M+15;    Use the serial line from the mini host
         EVENT: 36;               Send the XOFF

         DELAY: P(9,2);           Wait until the XOFF is completely sent

         BRANCH, 1:-              See if there are commands that need to be sent
                   -                    to the axis controllers
              IF, A(1) .GT. 0, OUTCOM:-     There are commands that need to be
                                  -    sent to the axis controllers
              ELSE, CHEKDONE;     There are no more commands to send to the axis
                            ;          controllers
```

```
;
;            OUTput the COMmands to the axis controllers
;
OUTCOM   DELAY: UN(13, 4);        Wait for the axes to become ready to receive

;            Send the commands to the axis controllers
         ASSIGN: X(36) = 3* 4*A(1) * P(9,3);     The transmission time:
         ;         3 axis controllers, 4 bytes/command, A(1) commands, time/char
         ASSIGN: X(37) = 0;       The commands are already set up, so don't
         ;                              store the message
         ASSIGN: X(38) = M+45;    The signal code to be sent
         ASSIGN: X(39) = 1;       Length of the message
         ASSIGN: X(50) = M+15;    Use the serial line from the mini host
         EVENT: 36;               Send the commands

         DELAY: 3 * 4*A(1) * P(9,3);   Wait until the data is sent

         BRANCH, 1:-              Check to see if the end of the command file
                 -                     has been reached
              IF, A(2) .EQ. 1, CHEKDONE:-   The end of the command file has been
                                -    reached
              ELSE, SENDXON;      There are more commands coming

;
;            CHEcK to see if the process is DONE
;
CHEKDONE BRANCH, 1:
              IF, X(M) .EQ. 0, PRODONE:-    The process is done
              ELSE, WAITDONE;     Wait until the process is done

;
;            WAIT until the process is DONE
;
WAITDONE RELEASE: MINIHOST(M);    Let the mini host do other things until the
                               ;         process is done

         QUEUE, 29;               Wait until the process is done
         WAIT: M+50;

         STATION, 7;              Entry point for the mini host polling entities
         ASSIGN: M = A(5);        Restore the station number

         QUEUE, M+60;             Get the mini host
         SEIZE: MINIHOST(M);

;
;            The PROcess is DONE
;
PRODONE  ASSIGN: A(6) = M+3;      Get the parameter set the range of values for
                            ;          the time it takes to bring the machine to
                            ;          its home position is stored
```

```
        DELAY: UN( A(6), 2);     Bring the machine home

;            Tell the system host that the machine is home - Send an SO
        ASSIGN: X(36) = P(9,2);  Set the transmission time
        ASSIGN: X(37) = M+30;    The queue the message goes to
        ASSIGN: X(38) = M+20;    The signal code to be sent
        ASSIGN: X(39) = 1;       Length of the message
        ASSIGN: X(40) = 15;      The message - an SO
        ASSIGN: X(50) = M+15;    Use the serial line from the mini host
        EVENT: 36;               Send the SO

        DELAY: P(9,2):-          Wait until the SO is completely sent

        NEXT( SENDXON );         Wait for another pallet

;
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                              ;
;                                                                              ;
;            The Axis Controller Model - Controlled by the axis                ;
;                 controller entity                                            ;
;                                                                              ;
;                                                                              ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;


;
;            The AXIS is DONE processing the input data
;
AXISDONE SIGNAL: M+50;           Tell the mini host that the axis is done
        ASSIGN: X(M) = 0;        Tell the world that the process is done
        RELEASE: AXISCONT(M);    Set the axis controller to idle

        QUEUE, 51;               Wait until new data is received
        WAIT: M+45;

        QUEUE, M+55;             Get the axis controller
        SEIZE: AXISCONT(M);

        ASSIGN: X(M) = 1;        Tell the world that the process is starting

;
;            CHecK to see if there is more DATA for the axis to process
;
CHK4DATA ASSIGN: X(37) = M+20;   Set up to read the queue where data is sent
        EVENT: 37;               Read the data (if there is any)

        BRANCH, 1:-              Check to see if there was any data
            IF, X(39) .EQ. 0, AXISDONE:-  There was no data, so the axis is
```

```
                                ¬    done
              ELSE, PROCDATA;     There was data, go and process it

;
;             PROCess the DATA input
;
PROCDATA DELAY: X(40):¬           Perform the command

         NEXT( CHK4DATA );        Go and check to see if there is more data

;
;
DISPOSE  ASSIGN: A(1) = A(1): DISPOSE; Get rid of anything that comes here
END;
```

F.3 FMS.EXP

```
BEGIN;
PROJECT, FMS MV1.23 EV1.13, Tim Martin,  5/ 7/85;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                    ;
;                                                                    ;
;                              FMS.EXP                               ;
;                                                                    ;
;                                 by                                 ;
;                                                                    ;
;                             Tim Martin                             ;
;                                                                    ;
;                    5/ 7/85              Ver 1.13                   ;
;                                                                    ;
;                                                                    ;
;        This experiment goes with the model that simulates the      ;
;            Virginia Tech FMS.  See FMS.DES for details.            ;
;                                                                    ;
;                                                                    ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;
DISCRETE,500, -The maximum number of concurrent entities
          13, -The maximum number of attributes per entity
          65, -The number of files
           8; -The number of stations
;
REPLICATE, 1, -The number of runs
    0, 7200;  -The run time - Base is 1 second
;
;
RESOURCES: 1-6, MACHINE,1,1: -Define the machines
           7,   SYSHOST:     -Define the system host
           8-12,  MINIHOST:-  Define the minihosts
           13-17, AXISCONT;   Define the axis controllers
;
TRANSPORTERS: 1, ROBOT,      -Transporter 1 is a robot
         1,                  -with capacity 1
         1,                  -using distance set 1
         0.608,              -velocity
         1-A;                -starting at station 1, ready for service -
                             -(Active)
;
;
DISTANCES: 1, 1-6,           -Distance set 1 for all stations (1-6)
                             -     Gives the distance between all stations
    10,20,30,20,10/          -Station 1 to station 2, 3, 4, 5 and 6
       10,20,30,20/          -Station 2 to station    3, 4, 5 and 6
          10,20,30/          -Station 3 to station       4, 5 and 6
             10,20/          -Station 4 to station          5 and 6
```

```
              10;          ~Station 5 to station              6
}
}
PARAMETERS:   1, 250:      ~Pallet arrival time mean
~
~       Cumulative probability that a pallet is of a certain type
             2, .25, 1,    ~25% are type 1
                .50, 2,    ~25% are type 2
                .75, 3,    ~25% are type 3
               1.00, 4:    ~25% are type 4
~
             3,   0,       ~The initial trace status ( 0 = No Trace,
                           ~    1 = Trace On )
               8000,8000,  ~The time when the trace status changes.
               8000,8000,  ~    There must be a change listed at or
               8000,8000,  ~    after the end of the run. There may
               8000,1.E9:  ~    be any number of changes during a run
~
             4,  23.25,    ~The pick time
                 22.5:     ~The place time
~
~       The minimum and maximum times it takes to get to a machine's home
~       position
             5,   0, 0:    ~The minimum and maximum time it takes to
                           ~    get to machine A's home position
                           ~
             6,   0, 0:    ~The minimum and maximum time it takes to
                           ~    get to machine B's home position
                           ~
             7,   0, 0:    ~The minimum and maximum time it takes to
                           ~    get to machine C's home position
                           ~
             8,   0, 0:    ~The minimum and maximum time it takes to
                           ~    get to machine D's home position
~
~       Transmission delay times
             9,   .0010,   ~The time it takes to transmit one character to the
                           ~system host from the input conveyor
                  .0010,   ~Character transmission time between the Mini-Host
                           ~and the system host
                  .0010:   ~Character transmission time between the mini host
                           ~and the axis controllers
~
~       Parameter set 10 contains the system host processing delays (including
~       disk access times)
            10, .0001,     ~The time it takes to check one element on the task
                           ~list
                .0002,     ~The time needed to transfer an entry in the input
                           ~queue to the task list
                .0116,     ~The time needed to set up for a robot move
                .0992,     ~The time needed to read one command from disk
```

```
                .0003:      ~The time needed to read one character in the
                            ~interrupt routine (The interrupt overhead is
                            ~included)
~
~       Parameter set 11 contains the mini host processing delays
            11, .385:       ~Percent of the minihost processing time needed for
                            ~the timer interrupt
~
~       Parameter set 12 contains the axis controller processing delays
~
~       The range of times until an axis will be ready to receive
            13, 0, 3:        ~Zero and the average maximum command time
~
~       Parameter sets 14-50 are currently unused
~
~       Define the machine sequences, processing times and the number of
~       commands on each machine for the different pallet types
            51,  2,  3,  4,  6,  0,   ~Machine sequence for pallet type 1
               190,120,240,  0,  0,   ~Processing times for pallet type 1
                36, 15, 55,  0:       ~Number of commands for each process
~                190 seconds on machine 2 done in 12 commands,120s and 10
~                 commands on machine 3, 240s and 36 commands on machine 4 and
~                then exit to machine 6 -the exit conveyor
            52,  2,  6,  0,  0,  0,   ~Machine sequence for pallet type 2
               190,  0,  0,  0,  0,   ~Processing times for pallet type 2
                45,  0:               ~Number of commands for each process
            53,  3,  6,  0,  0,  0,   ~Machine sequence for pallet type 3
               120,  0,  0,  0,  0,   ~Processing times for pallet type 3
                25,  0:               ~Number of commands for each process
            54,  4,  6,  0,  0,  0,   ~Machine sequence for pallet type 4
               240,  0,  0,  0,  0,   ~Processing times for pallet type 4
                80,  0;               ~Number of commands for each process
~       More pallet types can be added by changing the cumulative
~       probabilities of the different pallet types to include the new
~       pallet type(s), and then adding the machine sequence/processing
~       times/number of commands to the end of the parameter list.
;
;
RANKINGS: 1-9, LVF(1);  ~Process the pallets based in the order in which
~                        they came into the system.  (Order the entities
~                        in the queues based on low value of attribute 1
~                        first.)
;
;
ARRIVALS: 1, EVENT(98), 0, 1; ~Schedule the event that initializes the CPU
                              ~entities
;
;
INITIALIZE, X(7)=1.2;   ~The robot starts at station number 1
;
```

```
;
SEEDS:        ~Initialize the random number streams:
    1,374,:       ~Stream 1 is used for the time between pallet arrivals
    2, ,:         ~Stream 2 is used for the time it takes a machine to
                  ~    get to its home position
    3,766,:       ~Stream 3 is used to determine the part type
    4, ,;         ~Stream 4 is used to find the time until the axes become
                  ~ready to receive
;
;
COUNTERS: 1, PALLET NUMBER;  ~Counter number one keeps track of the number
                             ~    of pallets that have entered the system
;
TALLIES: 1, IN SYSTEM TIME;  ~For each finished pallet, find out how long
~                             it was in the system
;
;       Collect statistics on the following time varying parameters:
;            Output the changes to the file numbers specified - OUTPUT.nn
DSTAT:   1,  X(2), Station A Util., 20:
         2, NR(2), Station A Commit, 25:
         3,  X(3), Station B Util., 30:
         4, NR(3), Station B Commit, 35:
         5,  X(4), Station C Util., 40:
         6, NR(4), Station C Commit, 45:
         7,  X(5), Station D Util., 50:
         8, NR(5), Station D Commit, 55:
         9, NT(1), Robot Utilizatn, 60:
         10, X(6), # on Conveyor, 65:
         11, NQ(8), Task List Length:
         12, X(7), Robot Position, 70: ~Used only for plotting
         13, NR(7), System Host Util.:
         14, NR(8), Mini Host 1 Util.:
         15, NR(9), Mini Host 2 Util.:
         16, NR(10), Mini Host 3 Util.:
         17, NR(11), Mini Host 4 Util.:
         18, NR(12), Mini Host 5 Util.:
         19, X(11), Line to MiniHo 1:
         20, X(12), Line to MiniHo 2:
         21, X(13), Line to MiniHo 3:
         22, X(14), Line to MiniHo 4:
         23, X(15), Line to MiniHo 5:
         24, X(16), Line fr MiniHo 1:
         25, X(17), Line fr MiniHo 2:
         26, X(18), Line fr MiniHo 3:
         27, X(19), Line fr MiniHo 4:
         28, X(20), Line fr MiniHo 5;
;
;
;TRACE;
END;
```

Appendix G

FMS FORTRAN LISTINGS

G.1 EVENT.FOR

```
$STORAGE:2
C
C
C       EVENT.FOR
C
C       by
C
C       Tim Martin
C
C             5/ 7/85
C
C
C       This file contains FORTRAN subroutines for the EVENTs used in
C       the SIMAN programs HL and FMS, except for the trace events,
C       which are in the file TRACE.FOR.  The programs are described in
C       the file HL.DES and FMS.DES.  These routines must be linked
C       together with the SIMAN FORTRAN subroutines and the subroutines
C       in the files PRIME.FOR and TRACE.FOR to form a new SIMAN run
C       file.
C
C


      SUBROUTINE EVENT( PALLET, NEVENT )
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
C                                                                       C
C          EVENT calls the appropriate EVENT subroutine.  If            C
C          parameter 3,1 is zero and the event is a trace event         C
C          then no subroutine is called because the trace status        C
C          is off.                                                      C
C                                                                       C
C            12/21/84    Ver 1.10                                       C
C                                                                       C
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INTEGER PALLET, NEVENT


C          Check to see if this is event 11
```

```
      IF ( NEVENT .EQ. 11 ) THEN

C             It is event 11, the trace status is to change
          CALL CHGTR( PALLET )

C         Check to see if this is event 36
      ELSEIF ( NEVENT .EQ. 36 ) THEN

C             It is event 36, send the message requested
          CALL SNDMES( PALLET )

C         Check to see if this is event 37
      ELSEIF ( NEVENT .EQ. 37 ) THEN

C             It is event 37, receive the message sent
          CALL RCVMES( PALLET )

C         Check to see if this is event 50
      ELSEIF ( NEVENT .EQ. 50 ) THEN

C             It is event 50, update the task list
          CALL UPTASK( PALLET )

C         Check to see if this is event 51
      ELSEIF ( NEVENT .EQ. 51 ) THEN

C             It is event 51, print a communication error message
          CALL COMERR( PALLET )

C         Check to see if this is event 98
      ELSEIF ( NEVENT .EQ. 98 ) THEN

C             It is event 98, initialize the CPU entities
          CALL SETENT( PALLET )

C         Check to see if this is event 99
      ELSEIF ( NEVENT .EQ. 99 ) THEN

C             It is event 99, a message has arrived
          CALL MESARV( PALLET )

C         Check to see if the trace status is 'on' ( P(3,1) = 1 )
      ELSEIF ( P(3,1) .EQ. 1. ) THEN

C             The trace status is 'on' ( P(3,1) = 1 ), call the
C             appropriate trace routine
          CALL TRACE( NEVENT, PALLET )

      ENDIF

      RETURN
```

```
      END


      SUBROUTINE SNDMES( PALLET )
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
C                                                                       C
C     EVENT 36                                                          C
C                                                                       C
C     SNDMES sends a message.  The inputs are:                          C
C          X(36) is the transmission time                               C
C          X(37) holds the queue number the message is to be placed in  C
C                If X(37) is zero then no message is stored             C
C          X(38) is the signal code to be sent.  If X(38) is 0 then no  C
C                signal code is sent.                                   C
C          X(39) is the length of the message.  The maximum is 10.      C
C                This is unrelated to the transmission time as far      C
C                as this event is concerned. If X(39) is 0 then the     C
C                attributes of the invoking entity are used as a        C
C                message.                                               C
C          X(40) - X(49) is the message                                 C
C          X(50) is the transmission line to be used                    C
C     The outputs are:                                                  C
C          An event (#99) scheduled after the transmission time         C
C          The entity to arrive at event 99 is set up.  The first 10    C
C               attributes are the message.  Attribute 11 is the queue  C
C               the message is to be placed in and attribute 12 is the  C
C               signal code to be sent.                                 C
C                                                                       C
C     Ver:   1.03     5/ 7/85                                           C
C                                                                       C
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      COMMON/SIM/D(50),DL(50),S(50),SL(50),X(50),DTNOW,TNOW,TFIN,J,NRUN

      REAL A(15)

      INTEGER*2 PALLET, MESAGE, NA, LINE


C          Create a message
      CALL CREATE( MESAGE )

C          See if there is a message
      IF ( X(37) .NE. 0 ) THEN

C               Find out where to get the message from
          IF ( X(39) .EQ. 0 ) THEN
```

```
C                   Get the message from the attributes of the calling
C                   entity
                CALL COPY( PALLET, A )

           ELSE
C                   Get the message from X(40) up
                DO 10 NA = 1, INT( X(39) )
 10                 A(NA) = X(NA+39)

           ENDIF

      ENDIF

C          Set up to send the queue, signal code and transmission line used
      A(11) = X(37)
      A(12) = X(38)
      A(13) = X(50)

C          Store the message
      CALL ASSIGN( MESAGE, A )

C          Schedule the message to arrive after the transmission delay
      CALL SCHED( MESAGE, 99, X(36) )

C          Set the transmission line busy
      LINE = X(50)
      X(LINE) = X(LINE) + 1

C          If the line was already in use there is a Transmission ERRor
      IF ( X(LINE) .GT. 1 ) CALL TERR( LINE )

      RETURN

      END



      SUBROUTINE TERR( LINE )
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                           C
C                                                                           C
C     TERR prints an error message when a message is sent on                C
C     top of another message                                                C
C                                                                           C
C     Ver:   1.00     5/ 7/84                                               C
C                                                                           C
C                                                                           C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INTEGER*2 LINE
```

```
      WRITE(*, 100 ) LINE
  100 FORMAT('0', '**** Message on Line', I3, ' Overwriting Another',
     1     'Message ****' )

      RETURN

      END



      SUBROUTINE RCVMES( PALLET )
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C                                                                      C
C     EVENT 37                                                         C
C                                                                      C
C     RCVMES takes a message from the queue X(37) and stores it in     C
C     X(40) through X(49). It removes the message entity from the      C
C     queue and disposes of it. The length of the message is stored    C
C     in X(39) on exit. If there is no message X(39) is set to 0.      C
C                                                                      C
C     Ver:   1.02    12/20/84                                          C
C                                                                      C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      COMMON/SIM/D(50),DL(50),S(50),SL(50),X(50),DTNOW,TNOW,TFIN,J,NRUN

      INTEGER*2 MESAGE, NA


C          See if there is a message in the queue
      IF ( NQ(INT( X(37) )) .EQ. 0 ) THEN

C               There is no message
           X(39) = 0

      ELSE
C               There is a message, get it
           MESAGE = LFR( INT( X(37) ) )

C               Move the message to X(40) - X(49)
           DO 10 NA = 1, 10
                X(NA+39) = A(MESAGE, NA)
 10             IF ( X(NA+39) .NE. 0 ) X(39) = NA

C               Remove the message from the queue and dispose of it
           CALL REMOVE( MESAGE, INT( X(37) ) )
           CALL DISPOS( MESAGE )
```

```
      ENDIF

      RETURN

      END



      SUBROUTINE UPTASK( PALLET )
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                              C
C                                                                              C
C     EVENT 50                                                                 C
C                                                                              C
C     UPTASK UPdates the TASK list with the calling pallet.  If the            C
C     pallet is already on the task list, its entry is updated with            C
C     the attributes of the calling pallet.                                    C
C                                                                              C
C     Ver:   1.00     2/16/85                                                  C
C                                                                              C
C                                                                              C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INTEGER*2 PALLET, TASKLS, NPAL, NEWPAL

      REAL ATTRIB(15)


C          Find if the pallet is on the task list (Queue 8), and if it
C               is, where on the task list it is.  Use the pallet number
C               (A(7)) as a guide in finding the pallet.
      NPAL = A( PALLET, 7 )

      DO 10 TASKLS = 1, NQ(8)

C               Check to see if this entry on the task list corresponds
C               with the calling pallet
 10       IF ( INT( A( LRANK(TASKLS,8), 7 ) )  .EQ.  NPAL ) GOTO 20

C          The pallet is not on the task list
C          Put a copy of the pallet on the task list
      CALL CREATE( NEWPAL )
      CALL COPY( PALLET, ATTRIB )
      CALL ASSIGN( NEWPAL, ATTRIB )
      CALL INSERT( NEWPAL, 8 )

      RETURN

C          The pallet is already on the task list, update the task list
 20   CALL COPY( PALLET, ATTRIB )
      CALL ASSIGN( LRANK(TASKLS,8), ATTRIB )
```

```
      RETURN

      END



      SUBROUTINE COMERR( CPU )
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                             C
C                                                                             C
C     EVENT 51                                                                C
C                                                                             C
C     COMERR prints out an error message when there is a COMunication         C
C     ERRor when the mini host sends data to the system host.                 C
C                                                                             C
C     Ver:   1.00     2/16/85                                                 C
C                                                                             C
C                                                                             C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INTEGER*2 CPU


      WRITE(*,100) M( CPU )
 100  FORMAT( '0', '*****  Invalid Character Received by the System',
     1     ' Host  *****', /, ' ', '*****  Error on communication line',
     2     I2, '  *****' )

      RETURN

      END



      SUBROUTINE SETENT( ENTITY )
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                             C
C                                                                             C
C     EVENT 98                                                                C
C                                                                             C
C     SETENT SETs up the CPU ENTities for the FMS model.  An entity           C
C     is sent here at the beginning of each run.  This routine can            C
C     not be called from PRIME because this would cause HL to crash           C
C     when PRIME was called.                                                  C
C                                                                             C
C     Ver:   1.10     2/13/85                                                 C
C                                                                             C
C                                                                             C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

```
      INTEGER*2 ENTITY, MHN, CPU


C           Destroy the calling entity
      CALL DISPOS( ENTITY )

C           Create the system host polling entities
      CALL CREATE( CPU )
      CALL QUEUE( CPU, 11 )

C           Loop for the number of mini hosts (the number of machines)
      DO 10 MHN = 1, 5

C                Create system host interrupt routine entity
           CALL CREATE( CPU )
           CALL SETM( CPU, MHN )
           CALL QUEUE( CPU, 20 )

C                Create mini host polling entities
           CALL CREATE( CPU )
           CALL SETM( CPU, MHN )
           CALL SETA( CPU, 5, FLOAT( MHN ) )
           CALL ENTER( CPU, 7 )

C                Create axis controller polling entities
           CALL CREATE( CPU )
           CALL SETM( CPU, MHN )
 10        CALL QUEUE( CPU, 51 )

      RETURN

      END



      SUBROUTINE MESARV( MESAGE )
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                          C
C                                                                          C
C     EVENT 99                                                             C
C                                                                          C
C     MESARV stores a message sent in the appropriate queue and sends      C
C     the requested signal code.  The queue is stored in A(11) and the     C
C     signal code is stored in A(12).  If the signal code is 0 then no     C
C     signal code is sent and if the queue is 0 no message is stored.      C
C                                                                          C
C     Ver:   1.02     5/ 7/84                                              C
C                                                                          C
C                                                                          C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

```
      INTEGER*2 MESAGE, SIGCDE, LINE, QUEUE

      COMMON/SIM/D(50),DL(50),S(50),SL(50),X(50),DTNOW,TNOW,TFIN,J,NRUN

C           Send the signal code if requested
      SIGCDE = A( MESAGE, 12 )
      IF ( SIGCDE .NE. 0 ) CALL SIGNAL( SIGCDE )

C           Put the message in the requested queue
      QUEUE = A( MESAGE, 11 )
      IF ( QUEUE .NE. 0 ) CALL INSERT( MESAGE, QUEUE )

C           Reduce the transmission line's utility
      LINE = A( MESAGE, 13 )
      X(LINE) = X(LINE) - 1

      RETURN

      END
```

G.2 PRIME.FOR

```
$STORAGE:2
C
C
C    PRIME.FOR
C
C    by
C
C    Tim Martin
C
C     2/16/85
C
C    This file contains the routines that are run at the beginning of
C    every SIMAN run of HL or FMS.  It also contains the routine that
C    is called at the end of every SIMAN run.
C
C


      SUBROUTINE PRIME
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
C                                                                       C
C     PRIME calls the subroutines that initialize the system            C
C     before any SIMAN run.                                             C
C                                                                       C
C     Ver:   1.01    12/20/84                                           C
C                                                                       C
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C          Ask the user for any changes in the parameters
      CALL ASKP

C          Initialize the trace
      CALL SETTR

      RETURN

      END



      SUBROUTINE ASKP
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
C                                                                       C
C     ASKP is called before each run of HL and FMS. ASKP allows the     C
C     user to input any of the model parameters. The input is in        C
C     the form                                                          C
```

```
C          $1 $2 $3    or     0                                        C
C     In the first case, P( $1, $2 ) is set to $3. If zero is         C
C     entered the simulation is started.                              C
C                                                                     C
C     Ver:   1.01    12/20/84                                         C
C                                                                     C
C                                                                     C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      CHARACTER*80 LINE

      CHARACTER*1 CHR(80)

      INTEGER*2 PTR1, PTR2, PTR3, I, PSET, PARM

      REAL VALUE


C          Prompt the user for input and input a line
 10   WRITE(*,100)
 100  FORMAT( '0', 'Enter the parameter and new value ',
     1     '(0 to execute the simulation): ', \ )

      READ(*, '(A80)' ) LINE

C          Convert the input line into a character array
      READ( LINE, '(80A1)' ) ( CHR(I), I = 1, 80 )

C          Convert the first input value into a number
      CALL GETNUM( 1, CHR, VALUE, PTR1 )

C          Check for an invalid entry
      IF ( PTR1 .GT. 80 ) GOTO 10

C          Convert the input value to integer
      PSET = VALUE

C          Check to see if the user is done entering parameters
      IF ( PSET .EQ. 0 ) RETURN

C          Get the second input value, check for errors, convert to
C          integer
      CALL GETNUM( PTR1, CHR, VALUE, PTR2 )
      IF ( PTR2 .GT. 80 ) GOTO 10
      PARM = VALUE

C          Get the third number and check for errors
      CALL GETNUM( PTR2, CHR, VALUE, PTR3 )
      IF (PTR3 .GT. 80 ) GOTO 10

C          Set the parameter specified to the desired value
```

```
      CALL SETP( PSET, PARM, VALUE )

C           Go back and get more input
      GOTO 10


      END



      SUBROUTINE GETNUM( STPTR, CHR, VALUE, ENDPTR )
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                           C
C                                                                           C
C     GETNUM checks to see if there is a real number in the array CHR,      C
C     starting at or after position STPTR, and if there is returns the      C
C     number in VALUE. If there is not a real number in CHR, an error       C
C     message is printed and the routine returns ENDPTR as 81.              C
C     Normally, ENDPTR will be set to the position in CHR where the         C
C     decoded number ended.                                                 C
C                                                                           C
C     Ver:   1.01    12/20/84                                               C
C                                                                           C
C                                                                           C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      REAL VALUE

      INTEGER*2 STPTR, ENDPTR, PTR, PTR1, PTR2

      CHARACTER*1 CHR(80)

      CHARACTER*80 LINE


C           Find the first non-blank character
 10   DO 20 PTR = STPTR, 80
 20        IF ( CHR(PTR) .NE. ' ' ) GOTO 40

C           The line was blank, or there is an error in the number,
C           print an error message and return
 30   WRITE(*,*) 'Invalid Input!'

      ENDPTR = 81
      RETURN

C           Check for illegal characters until the next blank
 40   DO 50 PTR1 = PTR, 80
           IF ( CHR(PTR1) .EQ. ' ' ) GOTO 60
 50        IF ( ( ( CHR(PTR1) .LT. '0' ) .AND. ( CHR(PTR1) .NE. '.' ) )
     1          .OR. ( CHR(PTR1) .GT. '9' ) ) GOTO 30
```

```
C          Valid real number, convert from an ASCII array to an ASCII
C          variable
 60   WRITE( LINE, '(80A1)' ) '0', ( CHR(PTR2), PTR2 = PTR, PTR1-1 )

C          This line is needed (for an unknown reason!?!?) to make the
C          following line work
      WRITE( *,200)
 200  FORMAT( '+', ' ' )

C          Convert from an ASCII variable to a real number
      READ( LINE, * ) VALUE

      ENDPTR = PTR1
      RETURN

      END



      SUBROUTINE SETTR
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                          C
C                                                                          C
C     SETTR sets up the trace.                                             C
C                                                                          C
C     Ver:   1.01   12/20/84                                               C
C                                                                          C
C                                                                          C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      COMMON/SIM/D(50),DL(50),S(50),SL(50),X(50),DTNOW,TNOW,TFIN,J,NRUN

      INTEGER*2 TRTIME


C          Create an entity that will turn the trace on and off
      CALL CREATE( TRTIME )

C          Set the new entities' first attribute to point to the first
C          time the trace condition will change ( A(1) = 2 )
      CALL SETA( TRTIME, 1, 2. )

C          Schedule the next trace change event
      CALL SCHED( TRTIME, 11, P(3,2) )

      RETURN

      END
```

```
      SUBROUTINE WRAPUP
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C                                                                      C
C   This routine outputs a form feed to the screen before the SIMAN    C
C   summary report is output.  This puts the summary report on a       C
C   different page than any trace output.                              C
C                                                                      C
C    12/20/84    Ver 1.01                                              C
C                                                                      C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC


      WRITE(*, 100)
 100  FORMAT( '1' )

      RETURN

      END
```

G.3 TRACE.FOR

```
$STORAGE:2
C
C
C       TRACE.FOR
C
C       by
C
C       Tim Martin
C
C            2/16/85
C
C
C       This file contains FORTRAN subroutines for the trace EVENTs
C       used in the SIMAN programs HL and FMS. The programs are
C       described in the file HL.DES and FMS.DES. These routines must
C       be linked together with the SIMAN FORTRAN subroutines and the
C       FORTRAN subroutines in the files PRIME.FOR and EVENT.FOR  to
C       form a new SIMAN run file.
C
C


      SUBROUTINE TRACE( NTRACE, PALLET )
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
C                                                                       C
C       TRACE calls the appropriate trace printing subroutine.         C
C                                                                       C
C       Ver: 1.01     2/16/85                                           C
C                                                                       C
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INTEGER*2 NTRACE, PALLET


C
C          Call the appropriate trace routine
      GOTO (1, 2, 3, 4, 5, 6, 7, 8, 9, 10), NTRACE

C          Trace 1 is for when a pallet arrives to the system.
C          Used by HL.
 1    CALL ARRIVE( PALLET )
      RETURN

C          Trace 2 is for when a pallet is ready to move to another
C          machine (or the exit conveyor).
C          Used by HL.
 2    CALL MOVE( PALLET )
```

```
      RETURN

C           Trace 3 is used when a pallet gets use of the robot.
C           Used by HL.
 3    CALL ROBOT( PALLET )
      RETURN

C           Trace 4 is used when a pallet is done with the robot.
C           Used by HL only.
 4    CALL DWROBOT( PALLET )
      RETURN

C           Trace 5 is used when a pallet leaves the system.
C           Used by HL only.
 5    CALL FINISH( PALLET )
      RETURN

C           Trace 6 is for when a pallet arrives to the system.
C           Used by FMS.
 6    CALL FARRIV( PALLET )
      RETURN

C           Trace 7 is for when a pallet is ready to move to another
C           machine (or the exit conveyor).
C           Used by FMS.
 7    CALL FMOVE( PALLET )
      RETURN

C           Trace 8 is used when a pallet is done with the robot.
C           Used by FMS only.
 8    CALL FDWROB( PALLET )
      RETURN

C           Trace 9 is used when a pallet leaves the system.
C           Used by FMS only.
 9    CALL FFINIS( PALLET )
      RETURN

C           Trace 10 is used when a pallet gets use of the robot.
C           Used by FMS.
 10   CALL FROBOT( PALLET )
      RETURN

      END



      SUBROUTINE ARRIVE( PALLET )
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                             C
C                                                                             C
```

```
C          ARRIVE prints the trace information for a pallet that      C
C          has just entered the system.  Used by HL.                  C
C          Trace event 1.                                             C
C                                                                     C
C             2/16/85    Ver 2.03                                    C
C                                                                     C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      COMMON/SIM/D(50),DL(50),S(50),SL(50),X(50),DTNOW,TNOW,TFIN,J,NRUN

      INTEGER PALLET, NPALLT, PTYPE, NMACH

      NPALLT  = A( PALLET, 7 )
      PTYPE = A( PALLET, 2 )

      WRITE(*,100) TNOW, NPALLT, PTYPE-50, INT( P(PTYPE, 1) )
 100  FORMAT('0', 'Time:', F7.1, ' Pallet', I3,
     1 ' arrived.  Pallet type:', I3, ', Machine Sequence:  ', I1, \ )

      NMACH = 2
 10   IF ( P(PTYPE, NMACH-1) .EQ. 6 ) GOTO 20
      WRITE(*, 150) INT( P(PTYPE, NMACH) )
 150  FORMAT( '-', I1, \ )
      NMACH = NMACH + 1
      GOTO 10

C          Print the number of pallets waiting on the input conveyor
 20   WRITE(*,200) INT( X(1) )
 200  FORMAT( /, ' ', '               There are', I3,
     1 ' pallet(s) on the input conveyor' )

      RETURN

      END


      SUBROUTINE MOVE( PALLET )
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                     C
C                                                                     C
C          MOVE prints the trace information for a pallet that is     C
C          ready to move to another station or the output conveyor.   C
C          The robot or next station are not necessarily ready for    C
C          the pallet.  Used by HL.  Trace event 2.                   C
C                                                                     C
C            12/20/84    Ver 1.02                                   C
C                                                                     C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      COMMON/SIM/D(50),DL(50),S(50),SL(50),X(50),DTNOW,TNOW,TFIN,J,NRUN
```

```
      INTEGER PALLET, NPALLT, CMACH, NMACH

      NPALLT = A( PALLET, 7 )
      CMACH = A( PALLET, 3 )
      NMACH = A( PALLET, 8 )

      WRITE(*, 100) TNOW, NPALLT, CMACH, NMACH
  100 FORMAT( '0',  'Time:', F7.1, ' Pallet', I3,
     1 ' was done at Machine', I2, ', ready to move to Machine', I2 )

C          Print the machines that are unavailable
      CALL UNMACH

      RETURN

      END


      SUBROUTINE ROBOT( PALLET )
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                          C
C                                                                          C
C          ROBOT prints the trace information whenever a pallet            C
C          gets use of the robot.  This trace is output when the           C
C          robot is at the pallet's machine, not when the robot            C
C          starts moving to the pallet's machine.  Used by HL.             C
C          Trace event 3.                                                  C
C                                                                          C
C             2/16/85    Ver 1.03                                         C
C                                                                          C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      COMMON/SIM/D(50),DL(50),S(50),SL(50),X(50),DTNOW,TNOW,TFIN,J,NRUN

      INTEGER PALLET, NPALLT, NMACH

      NPALLT = A( PALLET, 7 )
      NMACH = A( PALLET, 8 )

      WRITE(*, 100 ) TNOW, NPALLT, NMACH
  100 FORMAT( '0', 'Time:', F7.1, ' Pallet', I3,
     1 ' was picked up by the robot, is on route to Machine', I2 )

      RETURN

      END


      SUBROUTINE DWROBOT( PALLET )
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                          C
```

```
C                                                                       C
C          DWROBOT prints the trace information when the robot is       C
C          done with a pallet at a machining station.  The processing   C
C          time for the pallet at the station is printed and a list     C
C          of pallets waiting for a machine or the robot is printed.    C
C          Used by HL.  Trace event 4.                                  C
C                                                                       C
C           12/20/84     Ver 1.02                                       C
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      COMMON/SIM/D(50),DL(50),S(50),SL(50),X(50),DTNOW,TNOW,TFIN,J,NRUN

      INTEGER PALLET, NPALLT, CMACH
      REAL PTIME

      NPALLT = A( PALLET, 7 )
      CMACH = A( PALLET, 3 )
      PTIME = A( PALLET, 4 )

      WRITE(*, 100 ) TNOW, NPALLT, CMACH, PTIME
 100  FORMAT( '0', 'Time:', F7.1, ' Pallet', I3,
     1 ' arrived at Machine', I2, ', Processing Time will be', F6.1,
     2 ' sec' )

C          Print which pallets are waiting for a machine or the robot
      CALL PWAIT

C          Print which machines are unavailable
      CALL UNMACH

      RETURN

      END


      SUBROUTINE FINISH( PALLET )
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
C                                                                       C
C          FINISH prints the trace information when a pallet is done    C
C          with the system.  It also prints a list of the pallets       C
C          that are waiting for a machine or the robot.  Used by HL.    C
C          Trace event 5.                                               C
C                                                                       C
C            12/20/84    Ver 1.02                                       C
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      COMMON/SIM/D(50),DL(50),S(50),SL(50),X(50),DTNOW,TNOW,TFIN,J,NRUN
```

```
      INTEGER PALLET, NPALLT
      REAL TSYS

      NPALLT = A( PALLET, 7 )
      TSYS = TNOW - A( PALLET, 1 )

      WRITE( *, 100 ) TNOW, NPALLT, TSYS
 100  FORMAT( '0', 'Time:', F7.1, ' Pallet', I3, ' left the system.'
     1 '  It was in the system', F7.1, ' sec.' )

C          Print which pallets are waiting for a machine or the robot
      CALL PWAIT

C          Print which machines are unavailable
      CALL UNMACH

      RETURN

      END



      SUBROUTINE FARRIV( PALLET )
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C                                                                      C
C          FARRIV prints the trace information for a pallet that       C
C          has just entered the system.  Used by FMS.                  C
C          Trace event 6.                                              C
C                                                                      C
C             2/16/85    Ver 1.00                                     C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      COMMON/SIM/D(50),DL(50),S(50),SL(50),X(50),DTNOW,TNOW,TFIN,J,NRUN

      INTEGER PALLET, NPALLT, PTYPE, NMACH

      NPALLT  = A( PALLET, 7 )
      PTYPE = A( PALLET, 2 )

      WRITE(*,100) TNOW, NPALLT, PTYPE-50, INT( P(PTYPE, 1) )
 100  FORMAT('0', 'Time:', F7.1, ' Pallet', I3,
     1 ' arrived.  Pallet type:', I3, ', Machine Sequence:  ', I1, \ )

      NMACH = 2
 10   IF ( P(PTYPE, NMACH-1) .EQ. 6 ) GOTO 20
      WRITE(*, 150) INT( P(PTYPE, NMACH) )
 150  FORMAT( '-', I1, \ )
      NMACH = NMACH + 1
      GOTO 10
```

```
C          Print the number of pallets waiting on the input conveyor
 20   WRITE(*,200) INT( X(6) )
 200  FORMAT( /, ' ', '               There are', I3,
     1 ' pallet(s) on the input conveyor' )

      RETURN

      END


      SUBROUTINE FMOVE( PALLET )
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
C                                                                       C
C          FMOVE prints the trace information for a pallet that is      C
C          ready to move to another station or the output conveyor.     C
C          The robot or next station are not necessarily ready for      C
C          the pallet.  Used by FMS.  Trace event 7.                    C
C                                                                       C
C            12/20/84    Ver 1.01                                       C
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      COMMON/SIM/D(50),DL(50),S(50),SL(50),X(50),DTNOW,TNOW,TFIN,J,NRUN

      INTEGER PALLET, NPALLT, CMACH, NMACH

      NPALLT = A( PALLET, 7 )
      CMACH = A( PALLET, 3 )
      NMACH = P( INT( A(PALLET, 2) ), INT( A(PALLET, 5) + 1 ) )

      WRITE(*, 100) TNOW, NPALLT, CMACH, NMACH
 100  FORMAT( '0',  'Time:', F7.1, ' Pallet', I3,
     1 ' was done at Machine', I2, ', ready to move to Machine', I2 )

C          Print the machines that are unavailable
      CALL UNMACH

      RETURN

      END


      SUBROUTINE FDWROB( PALLET )
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
C                                                                       C
C          FDWROB prints the trace information when the robot is        C
C          done with a pallet at a machining station.  The processing   C
C          time for the pallet at the station is printed and a list     C
```

```
C          of pallets waiting for a machine or the robot is printed.   C
C          Used by FMS.  Trace event 8.                                C
C                                                                      C
C           12/20/84     Ver 1.01                                      C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      COMMON/SIM/D(50),DL(50),S(50),SL(50),X(50),DTNOW,TNOW,TFIN,J,NRUN

      INTEGER PALLET, NPALLT, CMACH
      REAL PTIME

      NPALLT = A( PALLET, 7 )
      CMACH = A( PALLET, 3 )
      PTIME = A( PALLET, 4 )

      WRITE(*, 100 ) TNOW, NPALLT, CMACH, PTIME
 100  FORMAT( '0', 'Time:', F7.1, ' Pallet', I3,
     1 ' arrived at Machine', I2, ', Processing Time will be', F6.1,
     2 ' sec' )

C          Print the task list
      CALL PTASK

C          Print which machines are unavailable
      CALL UNMACH

      RETURN

      END


      SUBROUTINE FFINIS( PALLET )
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C                                                                      C
C          FFINIS prints the trace information when a pallet is done   C
C          with the system.  It also prints a list of the pallets      C
C          that are waiting for a machine or the robot.  Used by FMS.  C
C          Trace event 9.                                              C
C                                                                      C
C            12/20/84    Ver 1.01                                      C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      COMMON/SIM/D(50),DL(50),S(50),SL(50),X(50),DTNOW,TNOW,TFIN,J,NRUN

      INTEGER PALLET, NPALLT
      REAL TSYS

      NPALLT = A( PALLET, 7 )
```

```
      TSYS = TNOW - A( PALLET, 1 )

      WRITE( *, 100 ) TNOW, NPALLT, TSYS
  100 FORMAT( '0', 'Time:', F7.1, ' Pallet', I3, ' left the system.'
     1 '  It was in the system', F7.1, ' sec.' )

C           Print the task list
      CALL PTASK

C           Print which machines are unavailable
      CALL UNMACH

      RETURN

      END


      SUBROUTINE FROBOT( PALLET )
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                          C
C                                                                          C
C           FROBOT prints the trace information whenever a pallet          C
C           gets use of the robot.  This trace is output when the          C
C           robot is at the pallet's machine, not when the robot           C
C           starts moving to the pallet's machine.  Used by FMS.           C
C           Trace event 10.                                                C
C                                                                          C
C              2/16/85    Ver 1.00                                         C
C                                                                          C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      COMMON/SIM/D(50),DL(50),S(50),SL(50),X(50),DTNOW,TNOW,TFIN,J,NRUN

      INTEGER PALLET, NPALLT, NMACH

      NPALLT = A( PALLET, 7 )
      NMACH = A( PALLET, 8 )

      WRITE(*, 100 ) TNOW, NPALLT, NMACH
  100 FORMAT( '0', 'Time:', F7.1, ' Pallet', I3,
     1 ' was picked up by the robot, is on route to Machine', I2 )

      RETURN

      END



      SUBROUTINE CHGTR( TRTIME )
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                          C
```

```
C                                                                       C
C     CHGTR is used to change the trace status for both HL and FMS.     C
C     Event 11.                                                         C
C                                                                       C
C     Ver:   1.04     2/16/85                                           C
C                                                                       C
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      COMMON/SIM/D(50),DL(50),S(50),SL(50),X(50),DTNOW,TNOW,TFIN,J,NRUN

      INTEGER*2 TRTIME


C          Switch the trace status
      IF ( P(3,1) .EQ. 0. ) THEN

C               Set the status to 1
          CALL SETP( 3, 1, 1.0 )

      ELSE

C               Set the status to 0
          CALL SETP( 3, 1, 0.0 )

      ENDIF

C          Increment the pointer to the next change time ( A(1) = A(1) + 1 )
      CALL SETA( TRTIME, 1, A( TRTIME, 1 )+1 )

C          Schedule the next trace change
      CALL SCHED( TRTIME, 11, P( 3, INT(A( TRTIME, 1 )) )-TNOW )

      RETURN

      END


      SUBROUTINE PWAIT
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
C                                                                       C
C          PWAIT prints a list of the pallets that are waiting for      C
C          a machine or the robot.  Used by HL.                         C
C                                                                       C
C            12/ 5/84    Ver 1.01                                       C
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INTEGER RANK, NPALLT, NMACH, PALLET
```

```
C          Print the heading
      WRITE( *, 100 )
 100  FORMAT( ' ', '              Pallets waiting for the robot or'
     1 ' a machine:' )

C          See if there are any pallets waiting
      IF ( NQ(8) .EQ. 0 ) THEN

C               If not, then print that there are none
           WRITE( *, 200 )
 200       FORMAT( ' ', '                    None' )

         ELSE
C               If there are pallets waiting print which ones
           DO 10 RANK = 1, NQ(8)
C                    For each pallet print a line

                PALLET = LRANK( RANK, 8 )
                NPALLT = A( PALLET, 7 )
                NMACH = A( PALLET, 8 )

                WRITE( *, 300 ) NPALLT, NMACH
 300            FORMAT( ' ', '                    Pallet', I3,
     1           ' waiting for machine', I2 )

 10             CONTINUE

      ENDIF

      RETURN

      END


      SUBROUTINE PTASK
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                           C
C                                                                           C
C          PTASK prints the task list of pallets in the system and          C
C          the pallets on the input queue.                                  C
C          Used by FMS.                                                     C
C                                                                           C
C            12/ 5/84    Ver 1.01                                         C
C                                                                           C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INTEGER RANK, NPALLT, NMACH, PALLET
      REAL STATUS

C          Print the heading
      WRITE( *, 100 )
```

```
 100  FORMAT( ' ', '              The Task List:' )

C          See if the task list is empty
      IF ( NQ(8) .EQ. 0 ) THEN

C               If it is, print that it is
           WRITE( *, 200 )
 200       FORMAT( ' ', '                    Empty' )

      ELSE
C               If the task list is not empty, print the status of each
C               pallet
           DO 10 RANK = 1, NQ(8)

C                    For each entry print a line
                PALLET = LRANK( RANK, 8 )
                NPALLT = A( PALLET, 7 )
                STATUS = A( PALLET, 9 )

C                    Check to find the pallet's status
                IF ( STATUS .EQ. 1 ) THEN

C                         The pallet is busy on a machine
                     NMACH = A( PALLET, 8 )
                     WRITE( *, 300 ) NPALLT, NMACH
 300                 FORMAT( ' ', '                    Pallet', I3,
     1                ' is busy on machine', I2 )

                ELSE
C                         The pallet is waiting for a machine
C                         Check the pallet's update status
                     IF ( STATUS .EQ. 0 ) THEN

C                              The pallet has not been updated
                          NMACH = P( INT( A(PALLET, 2) ),
     1                               INT( A(PALLET, 5) + 1 ) )

                     ELSE
C                              The pallet has been updated
                          NMACH = A( PALLET, 8 )

                     ENDIF

                     WRITE( *, 400 ) NPALLT, NMACH
 400                 FORMAT( ' ', '                    Pallet', I3,
     1                ' waiting for machine', I2 )

                ENDIF
 10             CONTINUE

      ENDIF
```

```
C          Print which pallets are in the input queue
      CALL PINQ

      RETURN

      END


      SUBROUTINE PINQ
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                              C
C                                                                              C
C     PINQ prints the pallet numbers of the pallets that have entries          C
C     in the input queue.                                                      C
C                                                                              C
C     Ver:   1.01    12/20/84                                                  C
C                                                                              C
C                                                                              C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INTEGER*2 RANK, PALLET, NPALLT


C          Print the heading
      WRITE( *, 100 )
 100  FORMAT( ' ', '               Pallets in the input queue: ', \ )

C          Check to see if there are any pallets in the input queue
      IF ( NQ(9) .EQ. 0 ) THEN

C              The input queue is empty
           WRITE(*,200)
 200       FORMAT( 'None' )

      ELSE
C              There is something in the input queue
C              Print the first pallet waiting
           PALLET = LRANK( 1, 9 )
           NPALLT = A( PALLET, 7 )
           WRITE(*,300) NPALLT
 300       FORMAT( I3, \ )

C              Print the remaining pallets
           DO 10 RANK = 2, NQ(9)

                PALLET = LRANK( RANK, 9 )
                NPALLT = A( PALLET, 7 )
 10             WRITE(*,350) NPALLT
 350            FORMAT( ',', I3, \ )
```

```
C               Finish the line off
            WRITE(*,*) ' '

      ENDIF

      RETURN

      END



      SUBROUTINE UNMACH
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                             C
C                                                                             C
C           UNMACH prints which machines are unavailable.                     C
C           Used by HL and FMS.                                               C
C                                                                             C
C             12/ 5/84    Ver 1.01                                          C
C                                                                             C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INTEGER NUNAVL, UNAVL(5), MACH

C           Find out which machines are unavailable
      NUNAVL = 0
      DO 10 MACH = 2, 6
           IF ( NR(MACH) .GT. 0 ) THEN
C                     If the machine is busy, save the machine number
                NUNAVL = NUNAVL + 1
                UNAVL(NUNAVL) = MACH
           ENDIF
 10   CONTINUE

      IF ( NUNAVL .EQ. 0 ) THEN

C               If all machines are available, print it!
           WRITE( *, 100 )
 100       FORMAT( ' ', '               All machines are available' )

         ELSE
C               Print which machines are unavailable
           WRITE( *, 200 ) ( UNAVL(MACH), MACH = 1, NUNAVL )
 200       FORMAT( ' ', '               Unavailable machines:', 5I3 )

      ENDIF

      RETURN

      END
```

Appendix H

SIMAN PREPROCESSOR

```
$STORAGE:2
C
C
C       SIMPRE.FOR
C
C       by
C
C       Tim Martin
C
C        2/ 8/85
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                             C
C                                                                             C
C        SIMPRE is a preprocessor for SIMAN files.  The main reason           C
C        for it's existence is to allow comments on ANY line of a             C
C        SIMAN file.  Any comment preceded by a tilde ( ~ ) will              C
C        be removed by this preprocessor.  In addition, normal                C
C        comments are also stripped off to reduce the size of the             C
C        file to be processed by SIMAN.  In addition to removing the          C
C        tilde comments, this program attempts to put any partial             C
C        statement on the same line.  This is done for two reasons.           C
C        One is to reduce the size of the file sent to SIMAN.  The            C
C        other, and more important reason, is that some SIMAN commands        C
C        do not work when they are split onto two lines. Examples             C
C        are the DISCRETE and REPLICATE elements in the experimental          C
C        frame.                                                               C
C                                                                             C
C                                                                             C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C
      CHARACTER*1 CHR, SAVE(74)
      INTEGER*2 ICHR, LLEN, POINTR, I, FLAG, NTABS

C
C        UNIT 3 IS THE INPUT FILE
C        UNIT 4 IS THE OUTPUT FILE
C
      OPEN( 3, FILE=' ', STATUS='OLD', FORM='BINARY' )
      OPEN( 4, FILE=' ', STATUS='NEW', FORM='BINARY' )
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C                                                                      C
C        This section of code is used to transfer data from the        C
C        input file to the output file.  It checks for semicolons,     C
C        commas, colons, line feeds, and end of file markers.          C
C        If any of these symbols are found the program will            C
C        branch to code to handle these special characters.            C
C        If none of these characters were found the program just       C
C        loops through this section of code until a special            C
C        character is found.                                           C
C                                                                      C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC


 5    LLEN = 0

C
C        This section of code removes lines which are blank or have
C        only comments on them.
C
 6    READ(3) CHR
      ICHR = ICHAR( CHR )

C        Expand tabs to 8 blanks
      IF ( ICHR .EQ. 9 ) THEN
         CHR = ' '
         ICHR = 32
         DO 9 NTABS = 1, 8
            LLEN = LLEN + 1
            SAVE(LLEN) = CHR
 9          CONTINUE

        ELSE
         LLEN = LLEN + 1
         SAVE(LLEN) = CHR
      ENDIF

C        Check for the end of the file
      IF ( ICHR .EQ. 26 ) GOTO 70
C        Check to see if the line is empty, or has only a comment.
C        To do this check to see if the first non-blank, non-tab character
C        is a semicolon, tilde or carriage return.
C
      IF ( ( ICHR .EQ. 59 ) .OR. ( ICHR .EQ. 126 ) .OR.
     1   ( ICHR .EQ. 13 ) ) THEN
C        Remove everything up to and including the line feed
 7       READ(3) CHR
         IF ( ICHAR( CHR ) .EQ. 10 ) GOTO 5
         GOTO 7
```

```
      ENDIF
C        Keep looping as long as there are leading blanks
      IF ( ICHR .EQ. 32 ) GOTO 6
C        If there are no more leading blanks, we must save the line
      DO 8 I = 1, LLEN
 8       WRITE(4) SAVE(I)

C
C        This is the main part of the input loop
C
 10   READ(3) CHR
 15   LLEN = LLEN + 1
      ICHR = ICHAR( CHR )

C        Check for a tab and convert it to blanks up to the next tab
C        stop (1+n*8)
      IF ( ICHR .EQ. 9 ) THEN
         CHR = ' '
         ICHR = 32
         L = INT( (LLEN-1)/8 ) * 8 + 8
         DO 16 I = LLEN+1, L
 16           WRITE(4) CHR
         LLEN = L
      ENDIF

C        Save the character
      WRITE(4) CHR

C        Check for semicolon
 17   IF ( ICHR .EQ. 59 ) GOTO 20
C        Check for comma, colon, slash
      IF ( ( ICHR .EQ. 44 ) .OR. ( ICHR .EQ. 58 ) .OR.
     1   ( ICHR .EQ. 47 ) ) GOTO 30
C        Check for end of line (line feed)
      IF ( ICHR .EQ. 10 ) GOTO 5
C        Otherwise, go get the next character on this line
      GOTO 10


CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                             C
C                                                                             C
C        This section of code removes everything after a semicolon            C
C        until a carriage return is found (until the end of the               C
C        line is found).  The carriage return is saved in the                 C
C        output file.                                                         C
C                                                                             C
C                                                                             C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

```
 20   READ(3) CHR
      IF ( ICHAR( CHR ) .EQ. 13 ) GOTO 15
      GOTO 20


CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                            C
C                                                                            C
C        This section of the program is used to remove any                   C
C        trailing blanks or tabs after a comma, colon or slash. It           C
C        checks to see if a tilde comment is present. If one is,             C
C        the program branches to the section of code that                    C
C        removes the comment.  This section of the program also              C
C        checks if the end of a line is reached.  If the end of              C
C        the line is reached, the program jumps to the section               C
C        of code that sees if two line can be concatenated.                  C
C                                                                            C
C                                                                            C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC


 30   READ(3) CHR
      ICHR = ICHAR( CHR )
C        Check for a space or tab
      IF ( ( ICHR .EQ. 32 ) .OR. ( ICHR .EQ. 9 ) ) GOTO 30
C        Check for a tilde
 35   IF ( ICHR .EQ. 126 ) GOTO 40
C        Check for the end of the line (carriage return)
      IF ( ICHR .EQ. 13 ) GOTO 50
C        Otherwise, go and save the character
      GOTO 15


CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                            C
C                                                                            C
C        This part of the program removes everything after a                 C
C        tilde until a carriage return is found.  When a carriage            C
C        return is found, the program falls through into the                 C
C        check to see if two lines can be concatenated.                      C
C                                                                            C
C                                                                            C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC


 40   READ(3) CHR
      IF ( ICHAR( CHR ) .NE. 13 ) GOTO 40


CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                            C
```

```
C                                                                       C
C       This section of code inputs and saves the second line           C
C       in an attempted concatenation.  The line is saved in            C
C       memory.  Some of the code above is repeated for the             C
C       input of the second line.  The code is modified to meet         C
C       the needs of this section of code.  The line is input           C
C       only as far as is needed to see if the lines can be             C
C       joined.  After the needed part of the line is input,            C
C       the program goes to the next section of code that decides       C
C       whether or not to join the lines.                               C
C                                                                       C
C       The variable FLAG is used to keep track of which mode           C
C       we were in when the end of the second line was reached.         C
C       If FLAG equals 0, then we were in normal input and save         C
C       when the end of the line was reached.  If FLAG equals 1,        C
C       then we were removing trailing blanks and tabs after a comma,   C
C       colon or slash.  After a decision is made about joining         C
C       the lines, FLAG is checked and the program goes to the          C
C       appropriate section of code in the main part of the             C
C       program above.                                                  C
C                                                                       C
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC


 50   FLAG = 1
C        Read the line feed
      READ(3) CHR
      READ(3) CHR
C        Check for the end of the file
      IF ( ICHAR( CHR ) .EQ. 26 ) THEN
         WRITE(4) CHAR(10)
         WRITE(4) CHAR(13)
         WRITE(4) CHR
         GOTO 70
      ENDIF
      POINTR = 1
C        We are still removing blanks and tabs after a comma, colon or slash
      GOTO 58

C
C        Input and save characters on the second line.  Check for the
C        end of the useful line and for commas, colons and slashes.
C        The end of the useful line is signified by a carriage return
C        or a semicolon.
C
 55   READ(3) CHR
 56   ICHR = ICHAR( CHR )

C        Change tabs to blanks
      IF ( ICHR .EQ. 9 ) THEN
```

```
         CHR = ' '
         ICHR = 32
         SAVE(POINTR) = CHR
         POINTER = POINTR + 1
         LLEN = LLEN + 1
      ENDIF

      SAVE(POINTR) = CHR
      POINTR = POINTR + 1
      LLEN = LLEN + 1

C        Check for the end of the useful line (check for semicolon
C        or carriage return)
      IF ( ( ICHR .EQ. 59 ) .OR. (ICHR .EQ. 13 ) ) GOTO 60
C        Check for no comma, colon or slash
      IF ( .NOT. ( ( ICHR .EQ. 44 ) .OR. ( ICHR .EQ. 58) .OR.
     1   ( ICHR .EQ. 47 ) ) ) GOTO 55
C
C        Remove any trailing blanks or tabs after a comma, colon or slash
C        Also check for the end of the useful line.
C
      FLAG = 1
 57   READ(3) CHR
 58   ICHR = ICHAR( CHR )
C        Check for blanks and tabs
      IF ( ( ICHR .EQ. 32 ) .OR. ( ICHR .EQ. 9 ) ) GOTO 57
C        Check for the end of the useful line (carriage return or tilde)
      IF ( ( ICHR .EQ. 13 ) .OR. ( ICHR .EQ. 126 ) ) GOTO 60
C        Otherwise, store the character
      FLAG = 0
      GOTO 56


CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                          C
C                                                                          C
C        The next section of code decides if the two lines can             C
C        be joined together.  They can be joined if the length             C
C        of the new line is less than or equal                             C
C        to 74 characters.  If the lines can be                            C
C        concatenated, the SAVE array in memory is                         C
C        copied to the output file.  The SAVE array contains the           C
C        second line without leading carriage return, line feed            C
C        or leading blanks.  If the lines can not be concatenated          C
C        the leading carriage return, line feed and  9 blanks              C
C        must be stored in the output file before the SAVE array           C
C        can be stored.  The blanks are needed if this is the              C
C        MODEL frame.  After saving the second line on disk,               C
C        control returns to the main portion of the program.               C
C        Where in the program to go is based on the state of FLAG.         C
C                                                                          C
```

```
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC


 60   IF ( LLEN .LE. 74 ) THEN
C        Combine the lines:  store SAVE in the output file
         DO 62 I = 1, POINTR-1
 62         WRITE(4) SAVE(I)
         IF ( FLAG .EQ. 1 ) GOTO 35
         GOTO 17
        ELSE
C        Don't combine the lines
         WRITE(4) CHAR(13)
         WRITE(4) CHAR(10)
         DO 67 I = 1, 9
 67         WRITE(4) CHAR(32)
         DO 68 I = 1, POINTR-1
 68         WRITE(4) SAVE(I)
         LLEN = POINTR + 8
         IF ( FLAG .EQ. 1 ) GOTO 35
         GOTO 17
      ENDIF


CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C                                                                      C
C        This last section closes the files and quits the              C
C        program.                                                      C
C                                                                      C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC


 70   CLOSE(3)
      CLOSE(4)

      STOP


      END
```

The vita has been removed
from the scanned document