Hardware-based Parallel Simulation of Flexible Manufacturing Systems

by

Dong Xu

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfill the requirement for the degree of

Doctor of Philosophy
in
Industrial and Systems Engineering

F. Frank Chen, Chair
Michael P. Deisenroth
John P. Shewchuk
Kimberly P. Ellis
Nathaniel J. Davis

July 16, 2001
Blacksburg, Virginia

Keywords: Hardware-based Parallel Simulation, Real-time Simulation, Rapid Simulation of Flexible Manufacturing Systems

Hardware-based Parallel Simulation of Flexible Manufacturing Systems

Dong Xu

(ABSTRACT)

This research explores a hardware-based parallel simulation mechanism that can dramatically improve the speed of simulating flexible manufacturing systems (FMS) by applying appropriate enabling hardware technologies. The hardware-based parallel simulation refers to running a simulation on a multi-microprocessor integrated circuit board, called the simulator, which is specifically designed for the purpose of simulating a specific FMS. The board is composed of a collection of micro-emulators capable of mimicking the operation of equipment in FMS such as machining centers, transporters, and load/unload stations.

To design possible architectures for the board, a mapping technology is applied by making use of the physical layout information of an FMS. Under such a mapping method, the simulation model is decomposed into a cluster of micro emulator on the board where each workstation is represented by one micro emulator. Three potential architectures for the proposed simulator, namely, the bus-based architecture, the shared-memory based architecture, and the parallel I/O port based architecture, are studied. To provide a suitable parallel computing platform, a prototype simulator based on the combination of the shared-memory and the parallel I/O port architecture is physically built.

Besides the development of the hardware simulator, a time scaling simulation method is also developed for execution on the proposed simulator. The method uses the on-board digital clock to synchronize the parallel simulation being performed on different microprocessors. The advantage of the time scaling technology is that the sequence of simulation events is sorted naturally in consistent with the real events. In this way, no entangled waiting is needed as in the conservative parallel simulation methods so as to reduce the synchronization overhead and the danger of having deadlock. Experiments on the prototype simulator show that the time scaling simulation method, combined with the unique hardware features of the FMS specific simulator, achieves a large speedup compared to conventional software-based simulation methods.

ACKNOWLEDGMENTS

The accomplishment of this research is firstly attributed to the firm support from my advisor, Dr. F Frank Chen. His invaluable advice and help are always there to bring me out of the darkness of failures and lead to the next bright success. I am also greatly impressed by his dedication and enthusiasm to a research career, and felt fortunate to have him as my major advisor.

Many thanks to other committee members, Dr. Michael P Deisenroth, Dr. John P. Shewchuk, Dr. Kimberly P. Ellis, and Dr. Nathaniel J. Davis. Their valuable comments and suggestions helped improve the quality of this research.

Special thanks go to my fellow research group members, Wei Tang, Ming Dong, Layne Liu, and Jack Chen. Many intensive discussions and warm debates with them about this project stimulated a lot of new ideas contributing to the development of this research.

Lastly, I would like to express a full heart of thankfulness to my wife, Xuejing Hao. Her great support and patience have enabled me to complete this long journey of fulfilling my dream.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

A flexible manufacturing system (FMS) is defined as a complex manufacturing system characterized by high automation and massive alternatives in parts processing, parts routing, tool delivery, and capacity planning. These alternatives forge the foundation for the flexibility of an FMS. On the other hand, they also make it extremely difficult for analytical methods to study FMS taking all these details into account to provide an optimal operating strategy. Hence, simulation methods, with the ability to consider dynamics and uncertainties of the system, become essential and cost effective tools to design and analyze flexible manufacturing systems. But when applying software-based simulation methods to a real-time shop floor control, conventional software-based simulation usually fails to generate valuable results in an adequate time frame when responding to dynamic changes in real shop floors. To truly realize full real-time simulation and facilitate on-line control of FMS, especially under situations when unexpected events occur, this research explores a hardware-based, FMS specific parallel simulation method, which takes advantage of implementing digital electronic technology directly into simulation in order to accelerate the execution of the simulation.

1.1 Motivation

Since the late seventies, flexible manufacturing systems have become increasingly popular in manufacturing fields in accordance with the developing concept of customized production [104]. Although the capability to manufacture a variety of different part types on a versatile automated system is attractive, the capital investment for such a system usually is very considerable. Hence, the design, scheduling, and control of an FMS is very critical so that production needs can be satisfied in an optimal way. These aspects of FMS are addressed by significant research efforts, using both analytical and other methods. Computer-based simulation is a cost effective way to design, schedule and control of flexible manufacturing systems (FMS). Real-time simulation has been studied on various aspects of an FMS, such as capacity planning, bottlenecks detection, and schedule evaluation. But the lack of a suitable hardware platform is

one of the important reasons prohibiting the implementation of theoretically plausible real-time simulation in the shop floor of flexible manufacturing systems. The speed of traditional software-based simulation is far from adequate in meeting the requirement of real-time response. Although a very powerful mainframe computer with an array of processors does have the capability of fast simulation, it is too expensive to be realistic for an FMS. With recent dramatic advancement of microprocessor and parallel computing technologies, a multi-microprocessor circuit board specifically designed for real-time simulation and control of an FMS can become a viable mechanism in real industrial production.

1.1.1 Why Simulation?

Scheduling and control play important roles in achieving goals of better product quality, lower production cost, shorter lead times, and higher flexibility. But, as flexible manufacturing systems become more and more complex, scheduling and control of such integrated systems present a series of problems and challenges. Moreover, the capability of real-time scheduling and control of FMS in response to the occurrence of unexpected events (e.g., machine breakdown or tool breakage) is very desirable for highly automated FMS. Traditionally, these problems are studied using mathematical programming methods [17], [18], [38]. These analytical methods usually can provide a valuable solution within seconds and models can be developed within hours when using a well-developed modeling package, such as CANQ, a software tool package for queuing network analysis [113]. But the validations of final solutions rely heavily on the assumptions made for analytical models. These assumptions usually cannot hold because of many practical constrains in scheduling and control on real shop floors. For example, it is very difficult to represent robot control and limited queue length in a queuing network analysis. Therefore, for such a highly coupled nonlinear system as FMS, simulation becomes an important way to design, schedule and control due to the lack of applicable analytical methods.

Existing research efforts in the field of on-line simulation for FMS design and control strategies have been attempted in the following directions: (1) parallel processing technologies that enable the consideration of real-time system behaviors, such as distributed or parallel discrete-event simulation [2], [9], [20], [33], [79]; (2) on-line intelligent shop scheduling using simulation combining with expert systems or neural network methods [3], [41]; (3) Petri net based on-line simulation and control [115]. As a result, simulation software packages have been

widely used in the design and analysis of manufacturing systems. This is mainly due to three important characteristics of simulation software packages: the capability of providing solutions for "What-if" questions, the capability of considering details into the model, and the friendly user interactive interface. These simulation packages, especially those specifically developed for the purpose of manufacturing simulation, such as ARENA, PROMODEL, WITNESS, etc., provide very powerful tools for simulation of FMS.

1.1.2 What is Hardware-based Parallel Simulation?

The hardware-based parallel simulation refers to the simulation procedure performed on an FMS specific microprocessor-based simulator [14]. The hardware-based simulator is essentially a multi-microprocessor-based digital circuit board which is composed of a collection of "micro emulator units" to mimic the complete set of machinery and equipment in an FMS such as machining centers, transporters (conveyor, robots and AGVs), and load/unload stations. Each micro emulator unit consists of a microprocessor with its supporting peripheral microchips for local memory and communication. The operating logic of the designated resource such as processing times for different parts, expected mean time between failure (MTBF), and mean time to repair (MTTR), can be captured via micro-programs running on each micro emulator unit. In this way, a printed circuit board (PCB) can be built to capture the total operating conditions of an FMS, and the completed PCB can therefore be viewed as a lithograph of a real flexible manufacturing system.

1.1.3 Why Hardware-based Parallel Simulation?

There are significant drawbacks to software-based system simulation. For example, it is difficult to decide how detailed a simulation model should be built. Building and validating a simulation model requires tremendous work and is usually not reusable. For the practical need of implementing real-time scheduling and control of FMS, the major shortcoming of software-based simulation is that the execution of a complex simulation model built by using software-based simulation packages is too time consuming. It may take minutes to execute a simulation run for an 8-hour production planning "look-ahead window". Therefore, such software simulation packages cannot really provide the capability of real-time simulation and control, which is especially important when system disturbance occurs and rapid simulation is desirable.

These disturbances include those unexpected events, such as rush orders, tool breakage, or machine breakdowns, and a software-based system simulation usually fails to dynamically respond to these exceptional events. A hardware-based parallel simulation method means using as much digital hardware (e.g., microprocessors or Field Programmable Gate Arrays) and advanced computing technologies (e.g., parallel processing) as possible to carry out a simulation run specifically for an FMS. With the dramatic technological advancement of multiprocessor and parallel processing, a hardware-based simulator might be viable and superior to its software-based counterpart with respect to real-time scheduling and control. By directly manipulating at the board level, the hardware-based simulator is expected to be able to perform faster simulation than software-based simulation, and to provide direct support for the real-time control of FMS. In addition, the hardware-based simulator can provide a suitable hardware platform for other theoretically plausible real-time simulation methods as well.

Although in general, the proposed hardware-based parallel simulation method can be applied to other types of manufacturing systems, for instance, job shops, or even to other simulation fields, such as simulation for highway network, a battlefield. An FMS provides a valuable test bed for the proposed method because it can help to disclose the benefits as well as the limitations of such a method with its complex operational conditions, such as real-time capability to handle part routing, rush orders, and machine breakdowns.

In summary, realizing simulation directly on a multi-microprocessor based hardware simulator provides a great potential in reducing the overall computational time and considerably accelerating a simulation study. The advantage of this hardware simulator over the software simulation method is mainly achieved by the combination effect of two features, namely, the simplified focus and the multi-microprocessor architecture.

- The simplified focus. Generally speaking, the reason that hardware-based parallel simulation can be superior to the software-based simulation in terms of speed of simulation is that the hardware-based parallel simulation makes use of more hardware features to simplify the simulation. The complex simulation procedure, such as generating random numbers, sorting the event list, advancing the simulation clock, collecting the statistical results, etc., can be divided into several portions, and some of them can be realized by using some sort of hardware. The reason why it is possible to divide a simulation procedure into portions is that

the simulator is built specifically for simulating one FMS. The basic idea to execute simulation in a fast way is to mimic what happens in the real world but in a much smaller time scale. The following describes a rough estimation about the speed of a hardware-based parallel simulation. Suppose generating one entry on a microprocessor needs 10 instructions, and this entry represents a time period in the real world of 1 minute. Currently, most microprocessors are able to execute one instruction within 2×10^{-8} seconds if running under 50M Hz clock. Therefore, it takes the microprocessor $30\times2\times10^{-8} = 6\times10^{-7}$ seconds to execute these 10 instructions, assuming in average each instruction needs 3 clock cycles. Under such a calculation, 1 minute in a real world can be represented by 0.6 μs on the microprocessor. Then for a 100-hour simulation time period, it only takes the microprocessor $6\times60\times100\times10^{-7}$= 3.6 ms to execute, plus some necessary computational time for statistic collecting and calculation, which takes most advanced microprocessors approximately the same time to execute the simulation. Overall, the computational time of one simulation run for a 100-hour production time can be safely estimated as within 0.1s by most microprocessors. For 10 replications, the total computational time will be around 1 second. This is much faster than the software simulation, which usually requires minutes to simulate the same scenario.

- The multi-microprocessor architecture. In the proposed hardware-based simulator, each microprocessor, with its supporting microchips, is dedicated to mimic one important component (e.g., a machining center or a load/unload workstation) in the real FMS. Thus the simulator is actually implemented in a multi-microprocessor architecture enabling a large amount of computations to be executed in parallel. First, all activities related to the workflow can be represented exactly as what happens in the real world because of this multi-microprocessor architecture. For example, while operation of part 1 on workstation 1 is being simulated by microprocessor 1, microprocessor 2 can simulate the processing of part 2 on workstation 2 at the same time, exactly as what should happen according to the schedule. Second, by directly implementing simulation at the board level, the large synchronization overhead encountered in the software-based parallel simulation can be avoided by using some special hardware synchronization mechanism to synchronize communications among microprocessors. Finally, the computational time needed to generate the simulation report can be processed in parallel as well, which can further reduce this portion of execution time.

1.2 Problem Statement

A flexible manufacturing system is usually considered as a computer controlled and integrated configuration of numerically controlled (NC) machining centers with automated material handling systems, comprising conveyors, robots, and/or automated guided vehicles (AGV) [102]. A typical FMS is illustrated in Figure 1.1. Combining the merits of job shop production and flow shop production, an FMS is most suitable for mid-volume and mid-variety customized production. Due to the high-level distributed processing and material flow, an FMS inherits complicated design, planning, and operational problems. Solving flexible manufacturing related problems is relatively complex compared to that of traditional manufacturing systems where lead times are longer, inventory levels are higher, and the utilization rate is much lower. In a common scope, unique features of an FMS can be summarized as follows: (1) each machine in an FMS is quite versatile and capable of performing many different operations; (2) the system can machine several part types simultaneously; (3) individual part may follow different routes through the system. These capabilities and planning options, while making the system flexible, increase the difficulty of decisions associated with all phases of the life cycle of an FMS.

Figure 1.1 A typical flexible manufacturing system (courtesy of General Dynamics, Fort Worth Division)

The problems associated with the life cycle of an FMS can be broadly classified into four categories: design, planning, scheduling, and control [25], [27], [49], [91]. Simulation has been extensively used in all four areas. This research mainly focuses on applying real-time simulation for shop floor control, that is, using simulation to evaluate the schedule and provide performance results as feedback in order to reschedule and control FMS with the consideration of machine breakdowns. The main objective of this research is to develop a multi-microprocessor-based simulator for the applications stated above. Because no similar past studies have been carried out on the same topic, no prior research can suggest the direction of this research. In this research, primary efforts are to realize a pilot hardware-based simulator for FMS. These efforts include exploring possible hardware technologies that can facilitate the simulation, testing digital signal processor technology and time scaling simulation methodologies relating to the construction of a prototype simulator, and doing simulation experiments on such a simulator.

1.3 Research Objectives

The objectives of this research can be summarized as follows:

- Exploring the basic concepts about the hardware-based parallel simulation. Because no prior attempts have been made for the hardware-based parallel simulation, the first objective of this research is to explore the feasibility of the hardware-based parallel simulation. Also, theoretical proof of the causality and validation of the hardware-based parallel simulation should be done before a simulator can be built. Basic concepts related to a hardware-based parallel simulation need to be defined. While some of them can be borrowed from software-based parallel simulation, new innovative thinking is still required.
- Developing a prototype simulator and a corresponding simulation mechanism on an integrated digital board. This includes the task to investigate current or near-term advanced technologies that can be implemented on such a board. The most difficult task is to build a prototype simulator. This research will first be carried out based on some off-the-shelf multiprocessor boards, with necessary modifications to develop the simulator. Besides construction of the simulator, how to execute simulation on such a simulator is also a big

challenge. It is expected that the simulation mechanism on such a hardware board will be different from traditional system simulation methods.

- Experiment on the prototype simulator, verify and evaluate the performance of this hardware-based simulator by comparing its performance to that of the software-based simulation.
- Identifying the limitations of hardware-based parallel simulation through this research.

A prototype hardware-based simulator will be built to mimic the laboratory FMS located in the Flexible Manufacturing Systems Laboratory in the Grado Department of Industrial and Systems Engineering, Virginia Polytechnic Institute and State University. Components of the laboratory FMS which will be considered in this research are the parts, workstations (machines, load/unload station), storages (index tables), transportation equipment (conveyors, robots) and the material flow, while ignoring other supplementary equipment, such as the gauging stations, the cleaning stations, the inspection stations, etc. For the function of planning and control, efforts will be focused on the capability of rapid simulation, specifically for the situation when machines break down.

Considering the potential of this research, if the idea of hardware-based parallel simulation can be realized, it will bring benefits in three major aspects. First, the simulator can meet the requirement for real-time simulation applications, performing either rapid, on-the-loop, or bi-directional simulation of the FMS, which finally improves total production. Second, the idea of hardware-based parallel simulation provides innovative thinking in simulation areas not only for flexible manufacturing systems, but also for other types of systems requiring real-time control in combination with simulation. That is, the idea can be extended to simulations in the fields other than manufacturing, such as simulation in supply chain management, communication network, traffic control, and battlefield. Finally, the simulator can provide a suitable parallel computing platform to implement other remarkable on-line simulation methods.

1.4 Organization

This dissertation is organized as follows. In Chapter 2, a survey of simulation applications in manufacturing, especially distributed and parallel simulation methods that are

similar and might be useful to the hardware-based parallel simulation are summarized. Chapter 3 introduces an example of software-based simulation study, which can be used as the counterpart to the hardware-based parallel simulation. Chapter 4 describes details of methodology and design of the hardware-based parallel simulation, including the mapping method to build the simulator, possible architectures, and the time scaling and event-driven method to execute the simulation. Available and potential hardware technologies that can be used for the simulator are also explored in this chapter. In Chapter 5, an implementation effort of hardware-based simulation is introduced. Details about building a 4-DSP board, programming on the board, and other related issues for implementation are discussed. In Chapter 6, experiments of simulation scenarios for 1-workstaion, 2-workstation, and 4-workstation manufacturing cells are designed and conducted on the prototype simulator. While results show the hardware-based parallel simulation is promising in speedup, they also disclose some serious limitations of such a method. The last chapter, Chapter 7 summarizes conclusions and contributions of this research, and identifies some future research directions.

CHAPTER 2

SIMULATION IN MANUFACTURING SYSTEMS

In this chapter, various simulation methods and their applications in manufacturing systems are reviewed. Although this research focuses on hardware-based parallel simulation, it is still inspirable by studying from a review on its software side counterpart. These simulation methods include various on-line real-time simulation frameworks using software simulation tools, the standard clock method, distributed discrete-event simulations, and conservative or optimistic parallel discrete-even simulations (PDES). Among them, distributed and parallel simulation methods are of most interest because they make use of multiprocessor with parallel processing technologies, which bear some similarity to the core technology for the hardware-based parallel simulation.

2.1 General Characteristics of a Typical FMS

Before attempting to model an FMS, a good understanding of its system characteristics is necessary. A flexible manufacturing system is a production system consisting of a group of identical and/or complementary numerically controlled machines connected through an automated transportation system. A dedicated computer, called the cell controller, controls all operations of an FMS. An FMS is capable of processing different types of products in an arbitrary sequence with negligible setup delays between operations. Typically, an FMS can produce over 20 different part types simultaneously in the system. Some large ones can produce over 100 part types if a proper tool management system is provided. An FMS is distinguished from other types of manufacturing systems by following characteristics: high degree of functional integration, arbitrary operational sequence with negligible setup delays, complex tool management, and complicated control software.

High Degree of Functional Integration. The integration of an FMS can be summarized in two aspects, namely, integration among machining centers, and integration between machines and transportation subsystem. The machining centers used in an FMS are usually very versatile

and capable of performing different manufacturing operations with different specifications. For example, a drilling machine in an FMS may be able to drill any holes, internal or external, with a diameter from 0.1 mm to 30 mm. These machining centers can direct access to a large set of tools (ranging from 20 to150) from its tool magazine for different tooling requirements. The automatic tool exchange also reduces the impact of missing tools on normal operations. The links between machines and transportation subsystem are also highly integrated by adopting automated material handling equipments, such as robots and AGVs. However, the high integration within an FMS makes it difficult to divide the FMS into submodels when applying parallel discrete event simulation for system modeling.

Arbitrary Operational Sequence With Negligible Setup Delays. In the situations that several machining centers are capable of performing the same kind of operation, there exist alternative paths to route parts, especially when optimal system performance of the FMS is pursued [11]. With a number of alternative paths available for a part, individual part may follow different paths to finish all necessary operations through the FMS. It is likely that the next part of the same type may be sent to a different machine to be processed even for the same operation. Therefore, it is really difficult for analytical models to analyze the performance of the FMS. On the contrary, simulation models are able to capture exact production routings of a part in the FMS under different situations.

Complex Tool Management. Tool management is so critical to an FMS that it requires a complex tool supply subsystem, similar to the material handling subsystem, in an FMS. The tool management transports, prepares, and stores tools required for the processing of workpieces between the local tool magazines and the central tool storage. Tools have to be exchanged when part types to be processed within a certain time period are changed, if tool requirements are different as well. Ideally, tool exchange should not interrupt the currently running process. If it is true that the tool exchange is negligible, it can be ignored in the simulation models. Unfortunately, in practice, the tool management may become the bottleneck of the whole FMS. In this research scope, tool management issues will not be addressed.

Complicated Control Software. The highly automated nature of the FMS is achieved by the control of a complicated software package running on a cell controller computer [24]. The control system supervises most of the activities within an FMS, for example, downloading NC

programs to machining centers, moving parts and tools around using transportation equipment (e.g. a robot or an AGV), synchronizing the connection between machine and transportation system, issuing commands to each individual machine. To communicate with the cell controller, all components in an FMS must be linked with the cell controller. The cell controller has to respond quickly, or in real-time, to the request from any equipment. The data collection and status check, often neglected in other conventional production systems, are indispensable requirements for the operation of an FMS.

All these characteristics, which form the foundation to the flexibility of an FMS, are at the same time making it very difficult to be formulated in analytical models. Thus, simulation is widely employed as a suitable tool to design, plan, schedule and test an FMS, especially to provide answers to "what-if" questions [39], [56], [64], [65].

2.2 General Simulation Applications in Manufacturing Systems

Generally speaking, software-based simulations, whether using a common purpose simulation language (e.g., SIMAN, SLAM, or GPSS) or a simulation tool package (e.g. ARENA, PROMODEL, WITNESS, etc.), tend to model the FMS as a set of interconnected queues, in which a workstation is represented by a single-stage service facility with an input/output queue. The material handling system is usually considered as a resource for which these workstations compete. The load/unload stations are the entrance and exit of the simulation model. Such a queuing network can be illustrated as in Figure 2.1.

In such a network of queues, parts are "customers". It is the dispatching rules in the production schedule that determine how to route a part to the next machine. Before moving parts, the transportation resource, either a robot or an AGV, is acquired. Parts are attached to this resource which moves along with its path network to the destination machine. Resources are released when parts arrive at their destinations and are available for the next movement. As soon as a workpiece has been fully processed, it exits the FMS immediately. Figure 2.2 shows the sequence of operations in a typical part flow within such a queuing network, with two processing operations. From the viewpoint of flow, a part is simulated as being either in waiting, transporting, or processing state in the system. Within an FMS, unlike other production systems,

the part might be transported to any capable workstation at some decision points depending on dispatching rules.

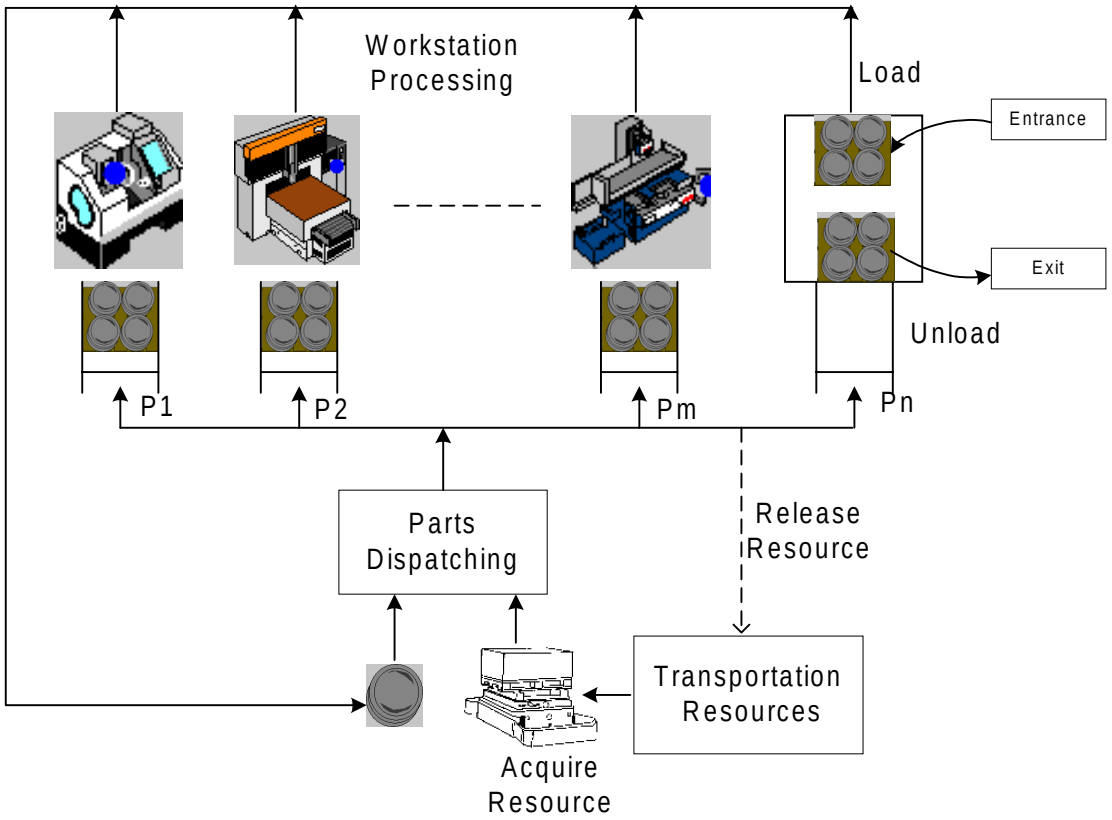


Figure 2.1 Queuing network based simulation model of the FMS

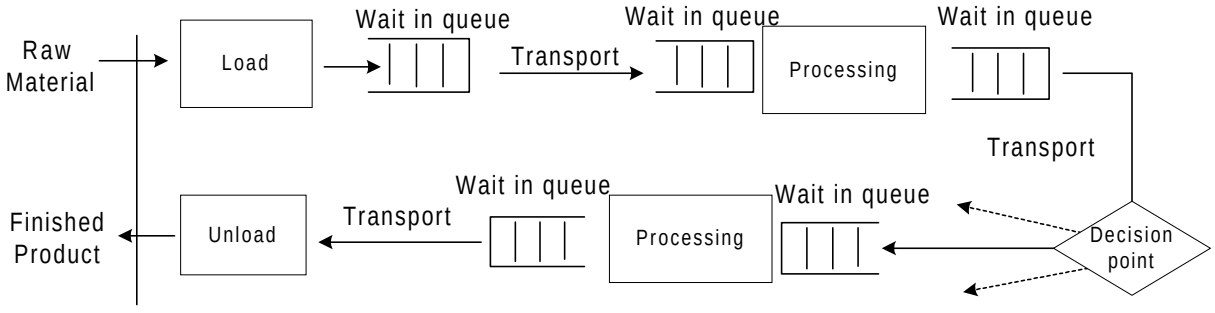


Figure 2.2 Part flow in a queuing network

2.2.1 On-line Simulation for Manufacturing Systems

Traditionally, simulation has been used for capacity planning, bottlenecks detection, and creating and testing manufacturing schedules [31], [67], [86]. Recent research efforts in this area appear to have shifted from off-line approaches to on-line approaches in two fields, one tends to focus on the models taking into account the real-time system behaviors, the other introduces the on-line intelligent methods combining the simulation especially for FMS scheduling.

In the real-time simulation of FMS, research efforts focus on developing methods to adjust the simulation model according to changing conditions on the shop floor. In the Discrete Event Dynamic System (DEDS) model, developed in 1996 by Fanti and et. al. [30], an abstract comprehensive framework for developing a generic control software for modeling the dynamics of a generic FMS at the job release and job flow levels was presented. The model was studied as an appropriate formalism that can lead to the capability of re-configuring itself as a consequence of internal and external changes. In the model, a 3-tuple sequential state C(t) at time t gave the information on system operating conditions while internal and external events serve as stimuli to the DEDS. The production planner served as the core of the model and was updated in real-time processing activity plans and scheduling policy, reacting to rapid changes in FMS operating conditions and management goals. Details of the real-time FMS operations were captured by the framework as well. In another research, a real-time software emulator for a Rapid Acquisition of Manufactured Parts (RAMP) flexible manufacturing system, the Hierarchical Object-Oriented Programmable Logic Simulator, was developed by Davis et. al. [24]. It employed a new simulation methodology to model the interactions among the coordinated object defined as the primary modeling element in the control architecture. The methodology also permits the immediate consideration of the detailed processing plan for the parts to be produced in the FMS. In this manner, the coordination of all entity flows, including the flows of the supporting resources (e.g. tooling, pallets, fixtures and processing plans), not only part flows were modeled through a stochastic queuing network representation of the FMS. Smith and Medeiros [96] described a method to separate the control strategy from the model of the physical system in order to achieve the goal to quickly and easily change a simulation model to incorporate the new policy. They demonstrated that separating the control strategy from the system description could be implemented in a conventional simulation language, thereby combining the benefits of flexibility with existing simulation language technology. This provided benefits to the designers

responsible for developing and implementing a FMS controller, as well as for users attempting to optimize system operations.

Another direction is to introduce intelligent methods into simulation to handle the complex decision-making tasks during the simulation. Gonzalez et al. [41] presented a typical viewpoint for on-line knowledge-based simulation for FMS. In his opinion, the knowledge-base on-line simulation system provided an effective approach to the decision-making problems in an FMS environment, and the ability to control the FMS environment by integrating several knowledge bases within a simulator and data communication software. This architecture was used for modeling, simulating, and analyzing several of the operational problems that characterize an FMS. A rule-based Expert System (ES) driven by a discrete event simulation model that performs dynamic shop scheduling is presented by Kunnathur and Sampath [58]. The following objective had been used in the ES shell: average flow-time, average tardiness, and percentage tardy. The entire rule based scheduling module was written in C and linked with SIMAN V runtime libraries to generate a final simulation module. Simple IF…THEN…ELSE rules were used by the system to schedule jobs. An intelligent simulation system (ISS) for FMS, developed by Baid and Nagarur [3], involved three levels: strategic planning, tactical level, and operation cell scheduling. The ISS had an intelligent front end (IFE) that can generate a simulation code, interacting with the user to obtain the requirements. The intelligent back end (IBE) component of the ISS took care of validation and statistical analysis and suggested different candidate models for evaluation. The heart of the ISS was the simulator that simulated the FMS and was developed using the simulation language SIMAN IV. The main feature of the ISS lies in the extension of the traditional simulation to intelligent simulation by the addition of IFE and IBE.

2.2.2 Simulation as Feedback to Scheduling of Manufacturing Systems

Another significant research direction was to integrate simulation methods with analytical methods for scheduling and control of FMS. The basic idea is using simulation as an evaluating tool to measure the performance of alternatives generated by scheduling algorithms and the simulation results are fed back to the scheduler for rescheduling. In this way, simulation functions as a feedback to the scheduling mechanism so as to form a closed-loop in generating better schedules. In addition, simulation also functions as an interface to the physical system to

capture the current status of the FMS. Duffie and Prabhu [28] presented a distributed scheduling method for heterarchical manufacturing systems using simulation to evaluate local schedules that were continually generated by local virtual controllers. The scheduling engine was composed of a loosely coupled, highly autonomous but cooperatively local scheduler. Local schedules were evaluated by a virtual simulation system. The virtual simulation system was directly linked to the real shop floor so that simulation was initialized with real system status before each run took place. Moreover, the simulation results were used as a feedback to a simple early-tardy heuristic, which formed a "closed-loop" intending to improve the global goal for the next scheduling effort. The authors suggested that an efficient distributed discrete event simulation could provide faster schedule evaluation, making it more practical in real-time. Kim and Kim [54] also developed a real-time scheduling method mainly based on a simulation mechanism and a real-time controller. In their method, simulation was used to evaluate different dispatching rules under the same conditions when a disruption occurs. Jeong and Kim [52] improved Kim and Kim's framework for the real-time scheduling by adding dynamic simulation models into the mechanism. The dynamic simulation model must deal with not only the current states of the system, but also the future uncertainties by generating the future disturbances according to a probability of future disturbances estimated from the past history. More simulation based scheduling methods can be found in the FMS scheduling review paper of Harmonosky [43], [44].

Usually simulation is regarded as an effective way to evaluate alternatives in the planning, scheduling, and control of FMS. Moreover, simulation is superior to Operation Research (OR) methods and Petri Nets (PN), especially in the capability of taking into account the details of a generic FMS. But simulation is often time-consuming in both model building and simulation running processes.

2.3 Parallel Simulation Methods and Their Applications in Manufacturing Systems

There are many attempts to improve the speed of simulation. The very basic one is the variance-reduction technique, which tries to reduce the variance of an output random variable of interest without disturbing its expectation, while achieving a desired precision with less simulation replications [62]. Other research efforts mainly focus on applying parallel computing

technology in the simulation. They can be categorized as either distributed simulation or parallel discrete-event simulation, originally developed by Chandy and Misra [12]. There are actually no explicit distinctions between distributed and parallel simulation. Sometimes, they are distinguished to emphasize the program granularity. That is, the distributed simulation refers to the running of different simulation programs on different machines, while the parallel simulation refers to the running of different processes of one simulation on different machines.

Most research efforts attempting to improve the simulation technology were carried out in the studies of parallel discrete-event simulation (PDES). The basic idea of PDES is the decomposition of a simulation model into submodels and distribute the simulation computations on different processors concurrently [36]. Processes communicate to each other by means of message passing. Different from conventional sequential discrete-event simulation, which utilizes a global clock variable to advance the simulation, there are no shared variables nor central control among processes in PDES [12]. Some general characteristics of PDES are as follows:

- The physical system being modeled is viewed as being composed of a number of interactive physical processes (PP). Each PP is represented by a logical process (LP). The interactions between PPs are represented by the message-passing channels between LPs. Thus, the simulator forms a queuing network composed of all the connected LPs.
- All actual interactive activities between PPs at a certain time are simulated by time-stamped event messages passing between their corresponding LPs. The time attached to the message stands for occurrence time of that event in the physical system.
- Each LP has its own associated local simulation clock. This local clock variable denotes how far the sequential portion of the simulation on this LP has been processed. LPs may advance the simulation to different points in time. By comparing the stamped time to its local clock, an LP is able to check the order of in-coming events from other LPs. If a violation of causality principle occurs, that is, an in-coming event happened earlier than the current local time, some correction mechanism must be employed in order to maintain the right sequence of events to be simulated and generate correct simulation results.

The effectiveness of the parallel simulation is largely affected by the decomposition methods applied to a simulation model. Overall, the decomposition methods suitable for the

parallel simulation can be classified as: multiple replication simulation, time segmentation simulation, and space parallel simulation [109].

2.3.1 Multiple Replication Simulation

An execution of a simulation model can be viewed as computing a sample path for a random process X = {x(t), 0≤ t ≤T or x(i), i=1, 2, …, N}. The simulation result is an estimation of the steady-state mean of state variable x as $E(x)=\frac{1}{T}\int_0^T x(t)d(t)$ or $E(x)=\frac{1}{N}\sum_{i=1}^{N}x(i)$, where T is the simulation time period of one sample and N is the total observation, respectively. Replications of such a simulation running are needed to meet the statistical sampling requirement. Then an intuitive parallel simulation approach is to run a replication, which is an execution of the whole simulation model, on each processor independently. When all runs are completed, the results from each replication are averaged together to obtain an estimate for the measure of interests. This method is referred to as the multiple replications [46]. It can be described as follow:

$$\overline{X}(P)=\frac{\sum_{j=1}^{P}E(x)_j}{P} \tag{2.1}$$

where: $\overline{X}(P)$, the final simulation results, average of replications
$E(x)_j$ the mean for replication j
P, total number of replications

If $\hat{X}(T)$ or $\hat{X}(N)$ is regarded as the "true" simulation result obtained from traditional sequential simulation, then $\overline{X}(P)$ is an unbiased estimation of $\{\hat{X}(T)$, if T$\rightarrow\infty\}$ or $\{\hat{X}(N)$, if N$\rightarrow\infty\}$. One problem of this method is the error brought in by the transient period. Each replication comes with its own transient period, which cannot be determined before simulation. Some methods, such as running a preview simulation to determine the transient period, can be used to estimate the transient period for each replication. Yet such kinds of estimations will bring in more deviations themselves. Generally speaking, multiple-replication simulation is appropriate only when the initial transient period is short.

Besides the effort made to run whole replications on each processor, there are methods developed to run portions of the replication on each processor. Standard Clock (SC) simulation is such a method implemented on multiple processors. The main theme behind SC simulation is that: discrete-event simulations with different parameters but similar structure share a great deal of commonality, and such commonality can be exploited and leveraged to maximize computational efficiency so as to minimize unnecessary duplications. The basic ideas of SC simulation are quite different from conventional sequential scheduling simulation. Traditionally, a simulation model is composed of two mechanisms, namely, the clock mechanism and the state-update mechanism. In traditional event driven simulation methods, the clock mechanism dynamically manages the event list, determines the next event time and type, and reports this information to the state update mechanism. The state is then updated and the new state information is reported back to the clock mechanism. The feedback of the new state information to the clock mechanism is critical because this information is indispensable for updating the event list. This procedure is shown in Figure 2.3.

In the Standard Clock method, however, no state information is necessary for updating the clock mechanism since the two mechanisms are completely decoupled [13]. As a result of this decoupling, the next event time and type generated by clock mechanism can be broadcast to more than one state update mechanism which represents discrete-event dynamic systems (DEDS) with similar structure but different design parameters or control policies. If the next event received by a special state update mechanism is considered impossible for the DEDS represented by this state update mechanism, this event will be ignored by this state update mechanism. Figure 2-4 shows the schematic view of SC method. There are three steps for a SC method:

Step 1: Generate a sequence of events (the event types yet to be determined). The interval time between two events is exponentially distributed with summation of rates of all events.

Step 2: The event type is determined through a Unif[0, 1] random number r for each event as follows: event type = 1, if $0 < r < A1/A$

2, if $Al/A \leq r < (A1 + A2)/A$

⋮

n, if $(A1 + A2 + \ldots + An - l)/A \leq r < 1$.

Step 3: Check event feasibility and construct a sample path.

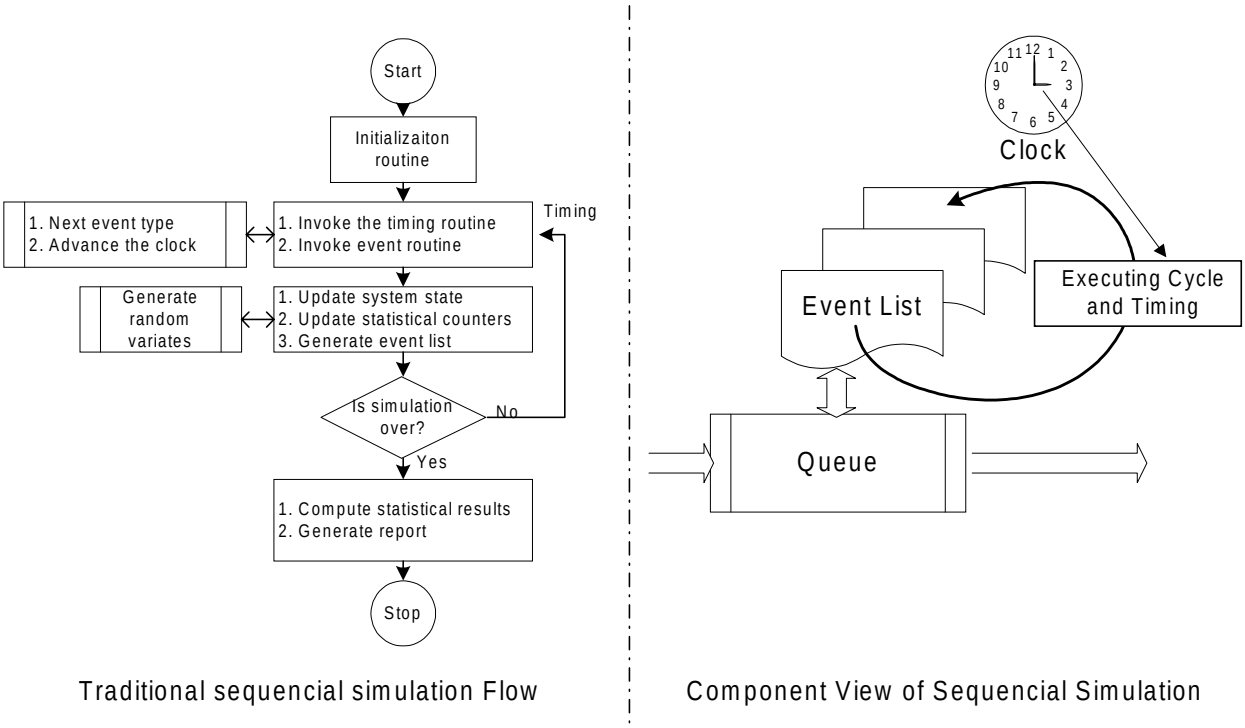


Figure 2.3 The clock and state-update mechanism of conventional sequential simulation

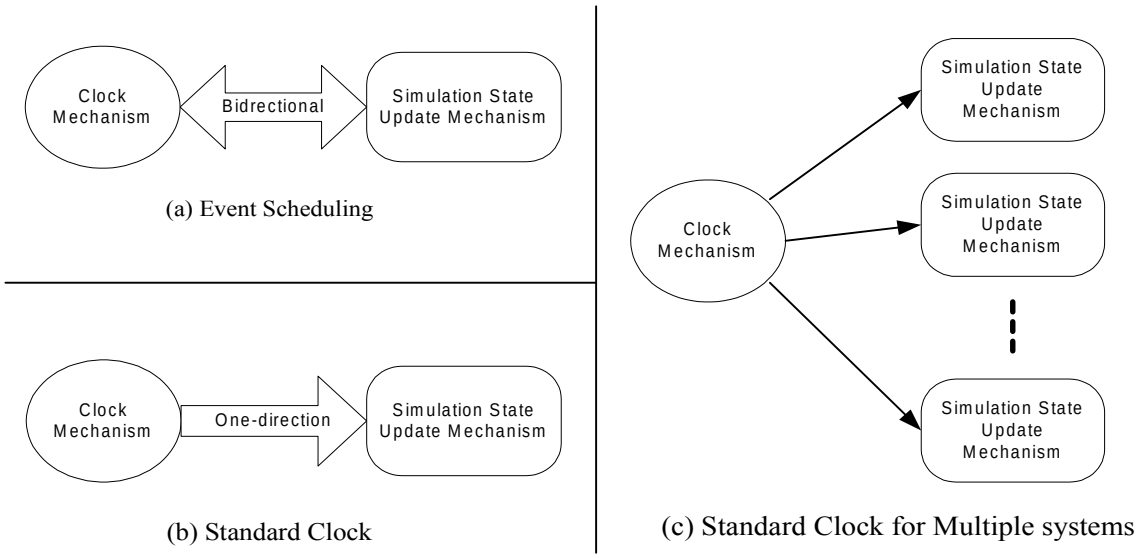


Figure 2.4 A schematic view of Standard Clock simulation method

The generation of event streams (Steps 1 and 2) is independent of system states, therefore, can be done off-line. Given an event stream, remaining task is to continuously check event feasibility (Step 3) during simulation. This significantly reduces the on-line simulation cost. Simplicity and ease of implementation are additional advantages of the SC method. When SC is applied to a set of parametrically different but structurally similar simulation experiments, the superiority of SC to conventional sequential simulation becomes even more significant. A very large speedup can be achieved in this way on a massively parallel SIMD machine [107]. In addition, when using a parallel computer, since the SC approach distributes the experiments over multiple processors, the problems of synchronization in the distribution of an inherently sequential simulation algorithm disappear.

2.3.2 Time Segmentation Simulation

Another idea of parallel simulation can be referred as time segmentation simulation. Basically, the simulation time period t is divided into several time intervals, and each time segment is simulated concurrently on one processor. This process is shown in Figure 2.5.

Because of its time-domain partitioning, time-parallel simulation can potentially yield massive parallelism [108]. However, to achieve the efficiency of the parallel simulation and at the same time generate accurate simulation results, the initial states for each time segment have to be accurately estimated. That is, an accurate prediction of states that occur in the future simulation time is required, which is theoretically impossible for general random processes. This limits the applicability of time-parallel simulation. Most existing time-parallel simulation algorithms are designed for specific models of memory-less processes, for instance, Markov Chains.

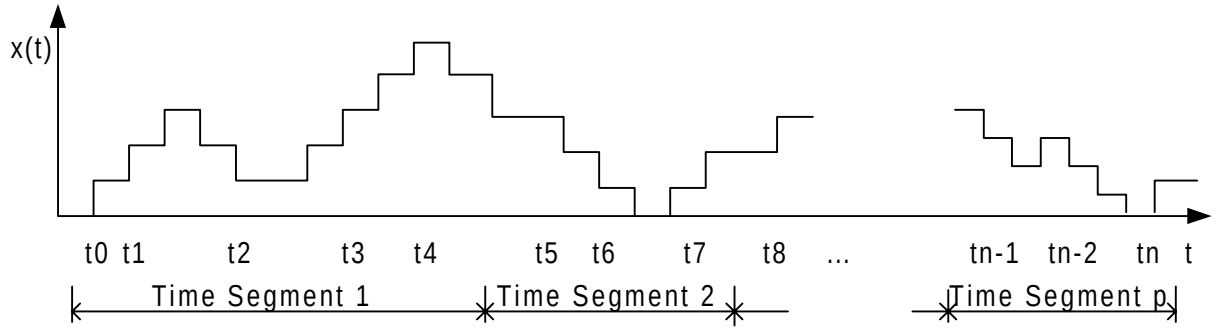


Figure 2.5 Time segmentation simulation

2.3.3 Space Parallel Simulation

Most parallel simulation methods can be classified into space parallel simulation, which basically decompose a simulation model into a number of components from the space domain of the model [76]. For example, a simulation model for an FMS can be divided into several submodels, in which each submodel simulates one part of the FMS, such as a machining center, a load/unload station, or a material handling unit. From the viewpoint of the simulation process, each component contains a disjoint subset of the model state variables. To execute the simulation, each component is mapped into a logical process (LP), which is responsible for computing the values of those state variables for the corresponding component over the simulation time period. Such an LP can be computed on one processor, or several LPs on one processor, determined by the possibility of parallelism of the simulation model. If all LPs run on only one processor, the problem is reduced into a conventional sequential simulation.

Preservation of the causality principles becomes a basic yet difficult problem for a successful parallel simulation, because each LP has its own local clock and those clocks may be advanced to different time points [23]. The causality principle states the fact that the past cannot be affected by the future. In the real world, if event A has some effect on event B, then event A must occur before event B. In other words, the cause must precede the effect. However, events having no direct or indirect relationships need not obey such sequencing constraints, and they need not follow the same time order. This nature allows parallel processing of these events. As a result, the simulation process on each LP is shortened.

In sequential simulation, time of the physical system is modeled as a global clock variable in the simulation program. Hence, causality can be easily ensured by ordering the events in increasing time sequence, and always advancing the global clock to the next smallest occurrence time. In parallel simulation, however, this single clock is replaced by a distributed clock mechanism. Each LP possesses and maintains its own local clock. For each LP, the partial ordering of the events can be ensured, but some kind of synchronization mechanism is needed to ensure that each event imposed by causality in the whole physical system are not violated. To meet such a requirement, two timing mechanisms, the conservative and the optimistic parallel simulation approach, were developed. The conservative approach strictly avoids causality errors at any time point during the simulation by implementing a global clock [36]. The global clock is

used to ensure that only event with the smallest occurrence time can be processed by its LP. During the simulation, the global clock waits until all directly or indirectly related activities before current time points are completed before moving to the next time point, thereby making the conservative approach become some sort of time-driven simulation. Although it is easy to maintain the causality principle in this way, the efficiency of parallel simulation is impaired because a remarkable computation time is wasted when most processors stay idle in waiting for the slowest event to be finished within one time step.

On the contrary, the optimistic approach allows temporary violation of causality and provides mechanisms to correct them. In this method, each LP possesses its own local clock and is responsible for advancing its local clock during the simulation. The local clock in LPs can be advanced at completely different rates, eliminating the idle time used to maintain the coherence of the incoming events. If violations are found later as new messages coming from other LPs, the simulation process on this LP must roll back to some previous good points and start from that point again in addition with the new incoming events. The Time Warp method, developed by D.R. Jefferson [51], is the first optimistic parallel simulation approach. Time warp means that time can go back. The basic concepts about Time Warp include virtual time, time-stamped messages, and global/local control.

1. Virtual Time

Virtual time is a global, one-dimensional, temporal coordinate system used to maintain synchronization. It may or may not have a connection with real time. Virtual time system is a distributed system executing in coordination with an imaginary virtual clock that ticks virtual time. Global Virtual Time (GVT) is the minimum time of all local virtual time. It is used as the time horizon that any event with virtual time less than GVT cannot be rolled back.

2. Time-stamped Messages

Time-stamped messages are messages attached with virtual time information for the purpose of synchronization. The format of such a time-stamped message usually represented by a 5-tuple data structure as [sender, send time, receiver, receive time, sign]. Anti-message is also a message of the same content message but opposite sign. To maintain correct causality, it is necessary and sufficient that on each logic process messages are handled in stamped times order.

At any given moment, some processes are allowed to be ahead in virtual time according to their local virtual clock. But when earlier messages come in, the local time needs to be rolled back to the global virtual time. Moreover, any activity related to the messages happened between this roll back time intervals must be cancelled by sending out corresponding anti-messages. In this way, local virtual time is warped back to the time of the in-coming event with an earlier occurrence time.

3. Control

There are two levels of control in any parallel simulations. They are global control and local control, respectively. Global control handles situations such as termination, error handling, and input and output of the simulation. Local Control is composed of a local virtual clock, a state queue, an input queue, and an output queue within a roll back mechanism, shown in Figure 2.6. If a message comes in with a less stamped times than the local virtual time, roll back fires. The local time is rolled back to the stamped time of the message, or to the global virtual time, whichever one is less. Anti-messages are sent out to cancel all related events that were sent out within the roll back time period. Anti-messages annihilate each other if they are both in the input queue. Otherwise, they just wait in the queue to be processed. Cultivated from the basic Time Warp mechanism, there are a lot of improvements on local control issues to reduce the roll back overhead or prevent infinite roll back, such as the mechanism to fire a roll back, the mechanism to cancel anti-messages, the management of previous state, etc.

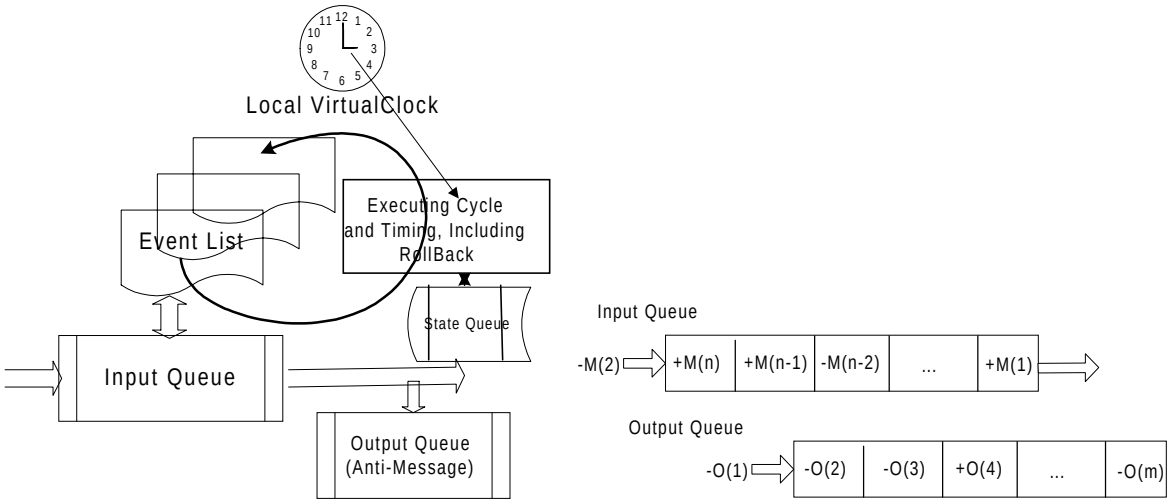


Figure 2.6 Component view of Time Warp local control

2.3.4 Distributed and Parallel Simulation in Manufacturing Systems

During the last decade, parallel simulation has found its use in many areas, such as simulation for digital logic circuit design, computer architecture, telecommunication network, transportation network, and manufacturing systems. The motivation of applying parallel simulation to the manufacturing systems mainly comes from the requirements from the real world to increase the computational performance. Comparing with the tremendous applications of parallel simulation in other areas (such as telecommunication and computer architecture), literature to date shows the research efforts made in applying parallel simulation for manufacturing systems are quite limited.

Early in 1985, Roberts and Shires [88] attempted to use the multiprocessing technique to further increase the speed of execution of manufacturing system simulation models. The initial configuration of the multiprocessing simulation system consisted of two single board computers, an SBC86/30 and an SBC86/12A, which were plugged into a Multibus backplane hosted by an Intel SYS310 computer. Simulation of particular types of entities was taken in a modular form. This meant while one task in the computer system was mimicking the actions of machines with tool changes, at the same time, another task might simulate the actions of buffers and storage devices, and still another task might simulate a material handling mechanism, such as robot and AGVs.

Bhuskute and Mize [9] presented a way to take advantage of structural factors of the manufacturing systems to achieve efficient methods of parallel simulation. For a deterministic routing manufacturing system, a basic strategy of parallel simulation consisted of creating several distinct submodels (each running on a single processor) from the manufacturing system routing structure. The flow of parts along with their routing created a flow of messages across the submodels. Simulation execution of each submodel running on a single processor had its own event calendar which simulated the events within the boundary of the submodel. An Intel iPSC/2 concurrent supercomputer was used for the parallel simulation with NX/2 operating system running on each processor.

Fujii et al. [34] conducted PDES in a factory-level simulation using time bucket (a variation of time warp) synchronization mechanism. In their research, a CIM system (virtual factory) was partitioned into 5 areas. Each area was modeled on one processor of a computer

network (Ethernet) composed by 6 SUN SPARC workstations. Since the dependence among areas was not as strong as that of the cell level, by assigning one area to one processor, simulation could be executed independently most of the time. A synchronization mechanism called the time bucket (a variation of Time Warp) method was proposed. The method tried to take advantage of known transportation delays among areas to define a bucket time whose period was the minimum time interval, during which all the local events might be safely processed. Basically, each processor simulated its assigned model for such a predetermined bucket time, and rolled back only to the end of a bucket time if some contradictions occurred during that time interval. Comparison of computational time clearly showed the superiority of this distributed simulation.

Later in 1999, Fujii, Kidani et al. [35] improved the simple time bucket method to a modified time bucket mechanism (phased time bucket), which was composed of two phases, a local area simulation phase and a transportation systems simulation phase. At the local area simulation phase, during one time slot, the processors only processed its local events and registered all the requests for transportation in its local event list. Then at the following transportation phase, which started at the end of the previous time slots, the transportation requests were processed and messages were exchanged among processors. In this way, the connections between areas were further decomposed, and large speedup was achieved for the simulation of the specific virtual manufacturing factory.

Lim and Low [66] discussed how to implement dispatching rules in the parallel simulation of manufacturing. In a parallel simulation, because different portions of the simulation system can be at different time points, the dispatching rules computation may be complex, especially those rules tied up with time. They provided a request-reply protocol to handle the remote variables reading and writing of the shared-state variables in parallel simulation. Their finding was that if read/ write events were treated just like any other events and handled in timestamp order, there was no need to implement an additional synchronization mechanism.

When applying parallel discrete event simulation for an FMS, parallel discrete event simulation may not be very suitable, mainly because those tight connections within an FMS

makes it difficult to divide the system into independent submodels. This disadvantage can be illustrated in Figure 2.7 by comparing a typical manufacturing system model to an FMS model.

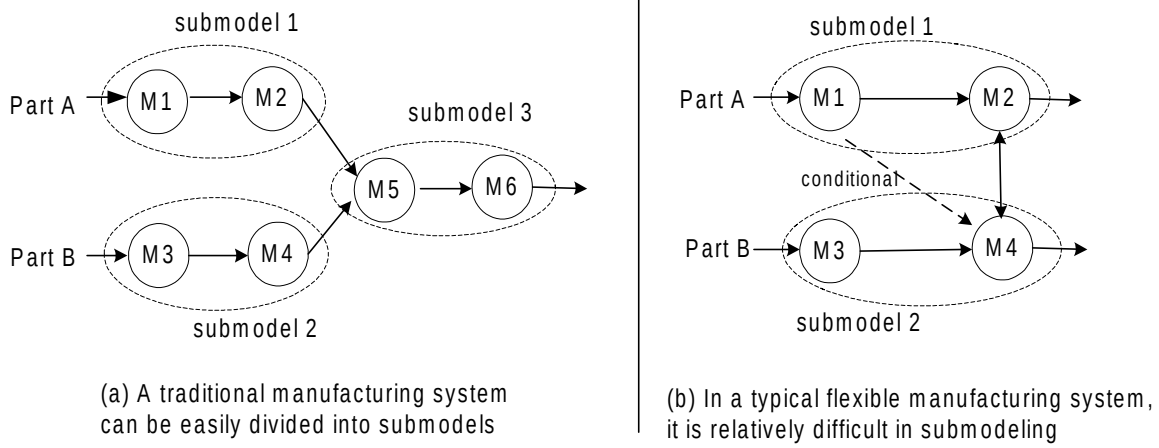


Figure 2.7 Comparison of decomposition model of traditional manufacturing systems and flexible manufacturing systems

The key point here is that besides the inputs and outputs between submodels, the processing within a submodel should be independent from other submodels. For example, in the traditional manufacturing system, the whole system can be easily divided into three submodels, submodel 1, 2 and 3, by grouping machines together. The independence between submodels means that when parts are routed from machine 1 to machine 2 in submodel 1, this activity will not affect the parts in submodel 2 to be routed from machine 3 to machine 4 at any time. Therefore, simulation process within individual submodel can safely advance concurrently, while overall process is still able to satisfy the causality constraints. However, in the FMS case, if partitioning the FMS into submodels in the same way as the traditional manufacturing systems, because there exists a conditional line between machine 1 in submodel 1 and machine 4 in submodel 2, a much more complex synchronization mechanism must be employed in order to safely advance the simulation process within each submodel in parallel. This conditional line is not uncommon in an FMS where real-time alternative routing paths are common in a product schedule. In the attempt to decompose the FMS into submodels, the great flexibility that allows parts to be routed to any possible machines can force the parallel simulation model to be divided

into submodels for individual machines. Under such a situation, a larger amount of roll back operations become frequent during the parallel simulation, which will dramatically increase the communication overhead and quickly degrade the efficiency of parallel discrete event simulation.

2.4 Hardware-Supported Parallel Simulation

With respect to microprocessor technologies, there are two types of microprocessors specifically designed for parallel computing. One is INMOS's transputer series (T805 as the typical one). The other is Texas Instrument's parallel digital signal processor (DSP) TMS320 C40. Most past research for hardware-support parallel simulation were conducted on transputers.

Pargas et al. [78] simulated an apparel manufacturing line on transputers. They broke the major functions of the simulation among the available processors. During simulation, one node transputer simulated one of several workstations and the root transputer took charge of the communication and command.

Hon and Ismail [48] also tried to use transputers to simulate manufacturing systems. They found out that although transputers can meet the basic requirement of parallelism, its low computing power cannot fulfill the requirement for complex computation tasks such as simulating a typical FMS. On the other hand, the TMS320 C40 parallel DSP show structures similar to those of a transputer. The performance, however, is on an average 10 times faster than that of a T805 in general mathematical calculations and extensive communication burden parallel computing [106]. Although no literature about applications of DSP in the simulation of manufacturing systems has been found so far, there are successful DSP applications in simulation and control of other dynamic systems [22], [83].

Table 2.1 summarizes the parallel simulation methods, hardware, and software used by researchers mentioned before. It shows that most researchers chose conventional local computer networks or existing general parallel computers, and not many researchers tried to implement parallel simulation directly at the board level. Some researchers chose transputer network as their primary developing platform. The problem with transputers is that although it can provide tight-

coupled communication links, it really cannot provide enough computing power for complex tasks required by parallel simulation.

Table 2.1 Summary of parallel simulation methods employed by researchers to date

Researcher (year)	**Simulation Method**	**Hardware Platform**	**Software**
Robert (1985)	N/A	An Intel SYS 310 connecting two single board computer through multibus backplane	Pascal
Paragas (1990)	Conservative	Transputer Network	Occam
Hon (1991)	Conservative	Transputer Network	Occam
Bhuskute (1993)	N/A	Intel IPS/2 concurrent computer with 32 nodes	Occam
Fujii (1994)	Time Bucket (optimistic)	Six Sun Sparc workstations connected through Ethernet	C
Fujii and Kidani (1999)	Phased Time Bucket (optimistic)	Six Sun Sparc workstations connected through Ethernet	C

Overall the survey shows that the success of parallel simulation in manufacturing systems depends on two key issues: model decomposition and synchronization among submodels. Model decomposition means how to break a simulation model into submodels, and assign them to different processes. Although a basic criterion is that decomposing the model in the way in which submodels are independent as much as possible, it is really application-specified. In other words, no general decomposition method that is suitable for all situations exists. For example, when a traditional manufacturing system can be easily divided into submodels by grouping workstations into cells, it is very difficult to decompose a modern manufacturing system, such as a flexible manufacturing system, where interconnections among the components are conditional and dynamic, as illustrated in Figure 2.7. The decomposition method used in most literatures is at the “cell level”, and some extend to the “factory level”. Moreover, it is not uncommon that the parallel simulation models of manufacturing systems are oversimplified to facilitate the decomposition. For example, in Bhuskute’s PDES model [9], the transportation subsystem was not considered and the process route for each kind of parts was assumed as deterministic. Oversimplified models usually cannot provide valuable information where detailed simulation for the system is required. The synchronization among submodels determines the efficiency of

parallel simulation, and it is a more hardware-related issue than a merely communication protocol problem.

The survey also shows that lack of a suitable hardware platform is another reason prohibiting the implementation of theoretically plausible parallel simulation methods for manufacturing systems. On the other hand, although a very powerful mainframe computer with an array of processors does have the capability for fast simulation, obviously it is too expensive to be an applicable solution on the shop floor of a manufacturing system. Nevertheless, with recent dramatic advancement of the microprocessor and parallel computing technologies, a multi-microprocessor based simulator which is specifically designed for parallel simulation of manufacturing systems can become a viable mechanism for realizing real-time parallel simulation and control of modern manufacturing systems in real industrial settings.

The thinking for using hardware to facilitate the simulation was stimulated by a study to obtain optimal routing paths for control of a maximum density automated storage and retrieval system (AS/RS) in an FMS via an analog resistance network [14]. In that research, unoccupied locations (AS/RS cells) were represented by a small resistance, while occupied locations were represented by a large resistance on a programmable digital circuit board. A heuristic algorithm enabling the search for the path of "least resistance" to the targeted location is the optimal path, thereby this hardware simulator of AS/RS provided an innovative technology for real-time simulation and control of maximum density AS/RS because the simulation can be completed within a negligible time interval. The potential expansion of this new simulation technology to the whole FMS led to the idea of developing a hardware-based simulator for flexible manufacturing systems.

Eventually, the idea of hardware-based parallel simulation for FMS really went through an evolution process. The very first thinking of using hardware to execute the simulation was stimulated by a study to obtain optimal routing paths for control of an automated storage and retrieval system (AS/RS) in FMS via an analog resistance network. The potential expansion of this resistance network to the whole FMS led to the idea of developing a hardware-based simulator for flexible manufacturing systems. While this original proposed concept seems exciting and feasible, initial investigations found out that such pure low level hardware implementation has difficulty in allowing us to timely configure parameters, collect output data,

and change the setup to reflect the true shop floor scenarios. Then, several multi-microprocessor based architectures were proposed to enhance the flexibility of the simulator itself. This revised concept simulator mainly consists of microprocessors, E^2PROMs, and dual-port SRAMs. Instead of resisters and capacitor network, E^2PROMs can be used for on-the-fly storing and changing of various simulation parameters during the simulation process. It needs to be tested that adding the programming capabilities to the hardware simulator will not hurt the speed of hardware-based parallel simulation too much. For future research, reconfigurable computing technologies can be adopted for further hardware-oriented simulator design to solve limitations with the multi-microprocessor design.

2.5 Detailed Research Steps

As explained in Chapter 1, the fundamental goal of this research is to make use of the continuously advancing digital electronic technologies and directly implement them in advanced simulation. To achieve this goal, the complete research endeavor can be divided into three phases, and each phase contains several detailed steps about what needs to be accomplished and the problems and difficulties associated with each step. The three phases are, namely, Phase one, feasible investigation of the hardware-based parallel simulation; Phase two, development of the simulator board and hardware-based parallel simulation methods; and Phase three, validation of the hardware-based parallel simulation. During the phase of feasible investigation, the feasibility of the hardware-based parallel simulation will be justified from both application and technical points of view. During the phase of development, a prototype of the simulator and corresponding hardware-based parallel simulation method will be developed. Based on experimental results obtained from this prototype, at the validation phase, the efficiency of the hardware-based parallel simulation will be justified. The tasks of these three phases can be further detailed into several steps as follows.

STEP 1: Exploring Basic Concepts of the Hardware-based Parallel Simulation

Because this research attempts to enter a totally new area in the simulation territory, even some basic concepts need to be carefully evaluated. The very first step of this research is to

distinguish the hardware-based parallel simulation from other simulation methods by answering following questions:

1) For basic concepts, what exactly does the hardware-based parallel simulation mean? What can be achieved by the hardware-based parallel simulation? Is the simulator simply something like the flight simulator or is there a difference in between?
2) From the technical point of view, is this hardware-based parallel simulation technically feasible? In other words, can it be realized according to current state-of-the-art technology and the foreseeable future technology?
3) From the application point of view, why can the hardware-based parallel simulation be superior to other simulation methods in the case of simulating FMS?
4) If the technology is feasible and beneficial, how should the hardware-based parallel simulation be implemented, by what kind of mechanism? Should we use a totally hardware wired digital board, or should we combine the hardware components with the software support?
5) Can we theoretically prove that the hardware-based parallel simulation method will not violate the causality principle, and how accurate are the results generated by the hardware-based parallel simulation?

Evidently, the attempt to answer these questions will lead the path of this research. Among all the research tasks, to develop the architecture of the simulator and corresponding hardware-based parallel simulation methodology is the most important part of this step. Basically, it is critical to investigate the suitable electronic technology that can facilitate the construction of the simulator, and develop a suitable simulation mechanism based on technical potential and the specific structure of FMS simulator to improve the speed of the simulation. The difficult point of this step is to fully investigate the feasibility and the research direction of the hardware-based parallel simulation. For example, among all the microprocessor technologies, such as transputers, FPGAs, general purpose microprocessors, microcontrollers, digital processing microprocessors, or some future single-chip multi-microprocessors, which one should be chosen, based on what kind of requirements, and how continuing advancement of microprocessor technologies can affect this research direction?

STEP 2: Software-based Simulation Study of the FMS

Before applying the hardware-based parallel simulation method to an FMS, a typical FMS must be chosen as the research subject, and its characteristics should be studied in depth. The study subject in this research is a laboratory FMS located at the Flexible Manufacturing Systems Laboratory of Virginia Polytechnic Institute and State University. Based on the physical features of the FMS, the simulation scenario must be designed to reflect the major characteristics of FMS simulation. Therefore two software-based simulation models are designed. One intends to simulate a basic level FMS operating, where the FMS behaves similarly to a production flow line. The other is a simulation of complex FMS operating under situations that uncertainties may occur (e.g., machine breakdown). Besides, performance parameters of major interest generated through the hardware-based parallel simulation methods can also be determined by studying the FMS.

The software-based simulation study of the FMS is also carried out as the counterpart of the hardware-based parallel simulation. The simulation results generated by the software-based simulation can be used as the benchmark to verify the results provided by the hardware-based parallel simulation. Moreover, through the software-based simulation, the inside execution mechanism of a simulation process will be explored, and factors affecting the speed of the simulation will be analyzed. This analysis can help identify those factors affecting the speed of the hardware-based parallel simulation and how these effects can be eliminated or reduced so as to improve the speed of the simulation.

STEP 3: Realization of a Simple Hardware-based Parallel Simulation

Because few literatures can be referenced and the investment to this research could be significant, the research is initiated by a simple trial effort to avoid any possible misleading. During this trial effort, the computing power of the microprocessors, the difficulty of communication between microprocessors, and the realization of a simple hardware-based parallel simulation is tested. At this step, two DSP starter kit (DSK) boards will be connected together through a dual-port SRAM so as to form a very simple simulator, on which the basic mechanism about the hardware-based parallel simulation can be tested. This simple test can provide a good understanding of the difficult points in realizing a full-scale simulator and its

corresponding simulation mechanism. It will also provide a reference guide for the purchase of hardware for a more powerful multi-microprocessor parallel development system.

STEP 4: Developing a Large-scale Simulation

The large-scale simulator is realized by either modifying suitable off-the-shelf multi-microprocessor development systems or expending the simple simulator to a large scale by adding more DSP boards if the simple one succeeds. No matter which way, the basic requirement for such a board is the same. That is, it must be a stand-alone multi-microprocessor development system with the capability of parallel processing. The board should have 4-8 microprocessors on board, with shared-memory and point-to-point communication links between any two microprocessors. For the software requirement, the capability of parallel debugging and compiling are highly desirable. To achieve the point-to-point communication, a dual-port SRAM between microprocessors might be necessary in case the number of on-chip communication ports of each microprocessor is not large enough to support all the required communication links. External memory will also need to be enlarged to store the huge amount of simulation records. Overall, this is the step that the simulator actually built.

STEP 5: Developing the Hardware-based Parallel Simulation Mechanism

In this step, the simulation mechanism developed at STEP 1 will be implemented on the simulator. The main tasks of this step are programming and debugging in assembly language. However, if a compiler which can automatically translate C code into desired assembly language code is available, the efficiency of the programming can be improved considerably. Even though, the assembly language code that is automatically generated by the advanced compiler still needed to be modified according to the developed hardware-based parallel simulation method. Some of the assembly language code, for example, the communication program, is better to be programmed by hand to favor the execution speed of the program. After the coding, the hardware-based parallel simulation can be executed to simulate the same scenarios as done by the software-based simulation study to obtain the simulation results.

STEP 6: Verifying the Hardware-based Parallel Simulation

Once the results are obtained from the hardware-based parallel simulation, comparisons can be made with those of the software-based simulation. If the results are close to each other,

they can be used to draw some conclusions for this research. If not, more effort is needed to find out hidden mistakes, correct them and run the new simulation again. Through this procedure, the hardware-based parallel simulation can be validated. Furthermore, the execution speed of the hardware-based parallel simulation will be recorded and compared to that of the software-based simulation, which will lead to the justification and conclusion of this research.

CHAPTER 3

A STUDY OF SOFTWARE-BASED SIMULATION

A software-based simulation study for the laboratory FMS can serve as a good reference to validate the hardware-based parallel simulation. In this chapter, a software-based simulation study for the laboratory FMS is introduced. Two working scenarios of the FMS, namely, a parallel production line and a manufacturing cell, were modeled. The software-based simulation models were built by using ProModel, a software simulation package specific for modeling manufacturing facilities. Performance parameters, such as throughput of the system, utilization of machines and transportation equipments, and mean time of parts flow were compared between these two models based on the simulation results generated by ProModel. Same procedure will be followed to build software simulation models for the same working scenarios as simulated by using the hardware-based parallel simulation. Comparison of the values of interested parameters generated by both software-based simulation and hardware-based parallel simulation can help to validate the hardware-based parallel simulation model.

3.1 The Laboratory Flexible Manufacturing System

To explicitly illustrate the idea of the hardware-based simulator, a laboratory FMS is used as the research subject. The laboratory houses a tabletop flexible manufacturing cell (see Figure 3.1) that consists of following components:

- 1 Pentium 166 PC as cell controller
- 2 CNC Lathes (Cyber Lathe)
- 2 CNC Mills (Cyber Mill)
- 2 material handling robots ("Puma" style, on linear travel slides)
- 3 index tables (WIP storage, microprocessor controlled)
- 1 four-camera machine vision system (with PX500 vision card)
- 4 belt conveyors (with built-in microcontrollers)
- 2 automatic part dispensers (computer controlled)
- 2 horizontal and 2 vertical gauging units (for height & width measurement)

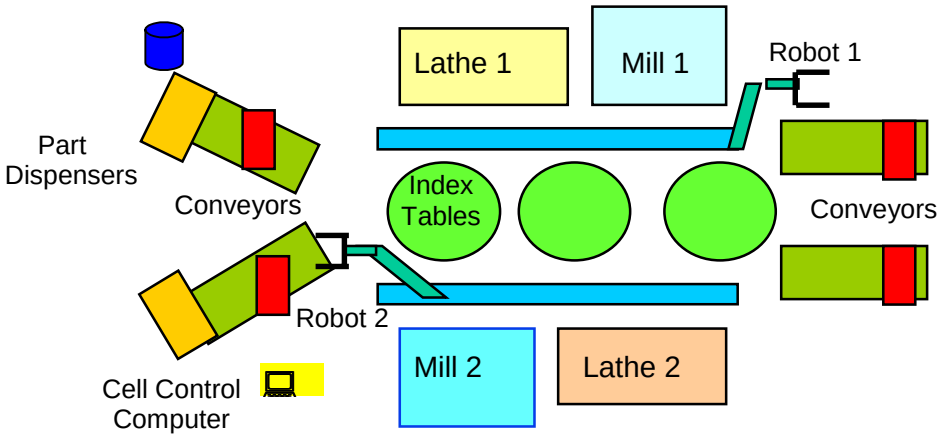


Figure 3.1 Layout of the laboratory FMS

The laboratory FMS uses a vendor supported control program, WALLI3, to manipulate the whole FMS, and IMPACT, an imaging processing software for the vision system.

One unique feature of the FMS is that the robots are seated on a slideway instead of a fixed location. The slideway provides an additional axis of movement to the robot so that each robot can move between two machines and is capable of performing part load/unload service to these two machines. Another special feature of the FMS lies in its symmetrical equipment layout. Along the central line, the machines on both sides are identical, with the ability to provide alternative part routing when the other is down. Hence, on the same side of the FMS, one cyber lathe and one cyber mill, along with the part handling robot serving them, naturally form a sub-level manufacturing cell. The existence of two identical sub-cells allows alternative part routing. Robots are not able to cross sub-cells, thus the index tables are used as the buffers to connect the two sub-cells, in addition to its basic function as the working-in-process storage. With the help of the index tables, the system enables more complex material flow within the FMS so as to better emulate real world manufacturing systems.

The basic operating sequence of the FMS is described below. Parts enter the FMS randomly from the two part dispensers and follow through the loading (input) conveyors connected to the dispensers. Attached to the conveyors are width gauges and height gauges which measure the dimensions of parts. If the dimensions of some parts exceed a specific range, they will be rejected and dumped to the bin at the end of the loading conveyors, and eventually

removed from the FMS. Otherwise, parts will wait at the end of the loading conveyors for a robot to carry them to dedicated machines to be processed. The robot performs the function as material handling. Thus, if a part needs further operation on another machine, the robot will pick up the in-progress part and deliver it to the destination machine. When all operations of a part are completed, the robot will pick up the part and send it to the inspection station where a computer vision system performs the quality inspection to decide if the part is to be accepted or rejected. Finally, accepted parts will be transported out of the FMS by one of two unloading conveyors, while the rejected parts will be dumped into bins attached to the other end of the unloading conveyors.

3.2 Introduction to ProModel

ProModel, developed by ProModel Inc., is the simulation package used to build software-based simulation models in this research. As a discrete event system simulator, ProModel primarily targets modeling discrete part manufacturing systems and has the advantages of ease-of-use, flexibility, modeling power, and realistic animation capabilities [7], [45]. ProModel's built-in constructs enable users to focus on manufacturing problems rather than on programming problems. Use of ProModel requires only a brief orientation and no programming skills. Most systems can be quickly modeled by simply dragging the built-in modeling elements to the right position. But external subroutines programmed by C or BASIC to implement complex logic can also be dynamically linked to the model. The modeling elements of ProModel provide fundamental building blocks to represent the physical and logical components of the system being modeled, such as parts, machines or resources. Major elements include locations, entities, path network, resources, processing, arrivals and other additional elements. ProModel allow users to focus on issues such as resource utilization, production capacity, system throughput, and inventory levels. The operation of an FMS is always defined freely by the designer, mostly through part flow, and operating logic and rules. During simulation, an animated representation of the system appears on the screen to monitor the current behaviors of the FMS. After a simulation run, performance reports about resource utilization, throughput rate, inventory levels, etc., are tabulated and graphed automatically for further analysis.

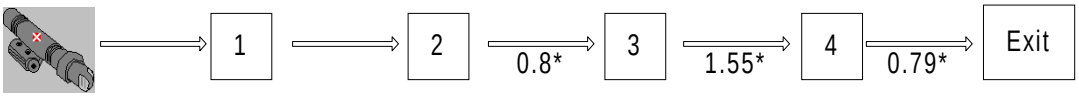
3.3 Simulation Study on the Flexible Manufacturing System Using ProModel

In this section, two hypothetical scenarios, the parallel production lines and the manufacturing cell, are designed so as to use simulation as the tool to analyze the effect of different decision policies upon the FMS. Criteria, such as throughout of the system, utilization of machines and equipment, etc., are compared between the two models based on the results produced by the simulation. Same procedure is followed later to develop all software-based simulation models that serve as the benchmark to the hardware-based parallel simulation models. Moreover, this study is also performed to disclose the difficulties of software-based simulation, and try to provide some useful hints to the proposed hardware-based parallel simulation method to overcome such difficulties.

3.3.1 Simulation of Parallel Production Lines

In the simulation model of the parallel production lines, the FMS is divided into two identical production lines by grouping machinery from each linear side of the FMS. Separated by the two robot sliding tracks, the model can be considered as two single flow lines working simultaneously. There are two parts, part A and part B, to be produced in the system. Part A needs both milling and turning operations, and is assigned to line 2, while part B needs only a milling operation and is allocated to line 1. Figure 3.2 and Figure 3.3 show the process plans for part A and part B, respectively. In both processes, operation No. 1 and No. 2 for Part A and Part B show the inspection procedure to ensure the quality of raw materials. When raw materials come into the FMS through the conveyor, the width and height gauges located on the loading conveyor measure the width and height of the raw material and send the information to the cell controller. On the cell controller, the control software compares them with some predetermined ranges. If either dimension of the raw material is out of a specified range, which means the material is not certified, it will be dumped and ceased from entering the FMS.

Machine breakdowns are also simulated in the model. In the real manufacturing systems, random downtime can result from such events as machine failure, part jams, and tool breakage. In the model, following assumptions are made about the downtime of machines. For Cyber Mill 1, the machine operates for an exponentially distributed time period with a mean of 600 minutes

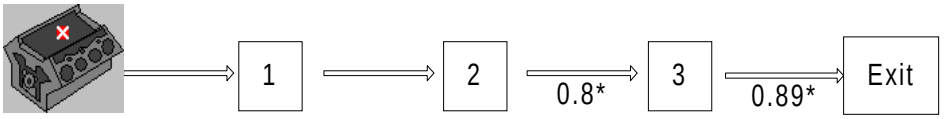


Part A

No.	Operation Description	Machine	Proc. Time (min.)
1	Width Measuring	Width Gauge	0.1
2	Height Measuring	Height Gauge	0.1
3	Face and straight turning	Cyber Lathe	N(6,0.1)**
4	Rough and finish milling	Cyber Mill	N(8,0.1)**

*: transportation time, including load/unload time, in min.
**: normal distribution

Figure 3.2 Process plan for part A on line 2



Part B

No.	Operation Description	Machine	Proc. Time (min.)
1	Width Measuring	Width Gauge	0.1
2	Height Measuring	Height Gauge	0.1
3	Rough and finish milling	Cyber Mill	N(4,0.1)**

*: transportation time, including load/unload time, in min.
**: normal distribution

Figure 3.3 Process plan for part B on line 1

before it breaks down. The repair time for the machine has a normal distribution with mean of 60 minutes and variance of 10 minutes. For Cyber Lathe 2, the machine operates for an exponential distribution work period with mean of 450 minutes before breakdown. The repair time for this machine also has a normal distribution with mean of 25 minutes and variance of 5 minutes. The

other machines are assumed never to break down in order to simplify the problem. Once the breakdown occurs, because of the single sequential workflow, the whole production line will stop until the broken machine is repaired and becomes operational again. Figure 3.4 shows such a situation during the simulation. When Cyber Lathe 2 breaks down, production line 2 stops. Although there is a part waiting to be delivered, the robot stops at its previous position until Cyber Lathe 2 becomes operational again.

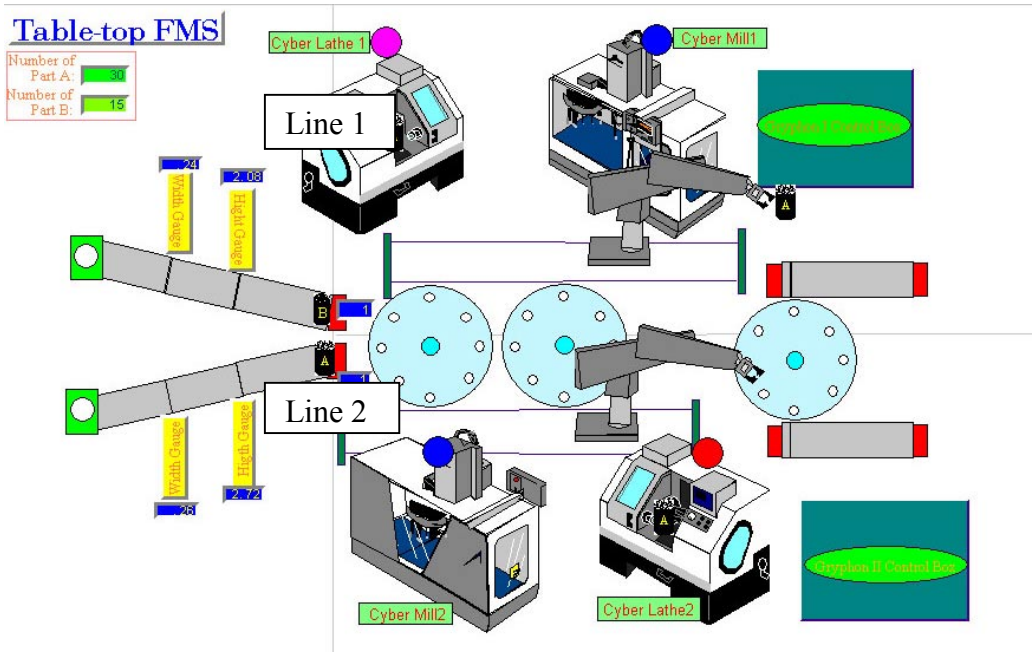


Figure 3.4 Cyber Lathe 2 breaks down and robot stops until the machine is repaired

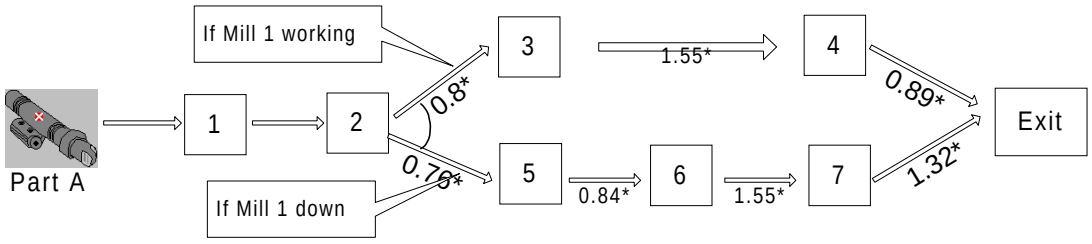
The simulation for the parallel production lines was carried out for 10 replications. Each replication ran first 16 hours as the warm-up period, and continued for the next 240 hours as the effective production period. The total execution time for 10 replications on a Pentium III 450MHz PC is 216 minutes with animation, and 66.1 seconds without animation.

3.3.2 Simulation of the Manufacturing Cell

When working as an integrated , flexible manufacturing cell, the FMS has more complex workflow within the cell than the single product flow line. For any part, there exist alternative flow paths, and a decision has to be made on-line to decide its next destination for the current part. Decisions are made according to either specific dispatching rules or customized control logic. In order to compare results with those of the production lines, the simulaton model of the manufacturing cell implement the same basic processes as the parallel production lines when no system disturbance (machine breakdown) occurs. But when machines break down, the manufacturing cell will deal with more complex situations than the parallel production line by deciding which alternative path a part should take.

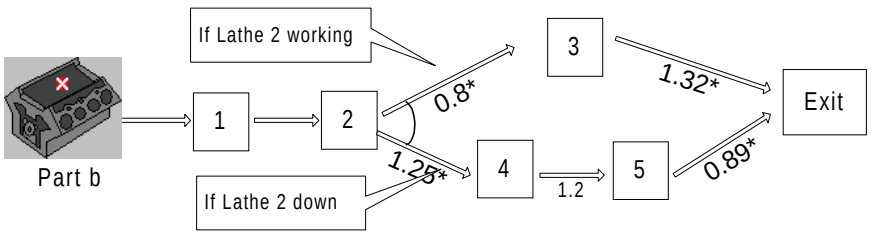
The control policy to handle the machine breakdown is set as: when a machine breaks down on one line, if the identical machine in the other line is operational, the affected parts will be delivered to the other line through the index tables. Under this decision policy, the new process plan for part A from dispenser 1 is shown in Figure 3.5. According to the plan, for part A coming from dispenser 1, if Cyber Mill1 is working, the part just follows the normal procedure on line 1; if Cyber Mill 1 is down, the part will be moved to the index table 1, and delievered to line 2 to finish whatever leftover operations until Cyber Mill 1 gets repaired. Then new arrivals of part A are processed back on line 1 again. For part A comeing from dispenser 2, if Cyber Lathe 2 is down, it will go through the index table 1 to line 1, and continue its leftover operations on Cyber Lathe1 and Cyber Mill 1. If both Cyber Mill 1 and Cyber Lathe 2 are down, the parts will be planced on the available position on the index table, and wait there until the machines return to operation.

For Part B, because the breakdown of Cyber Lathe 2 does not affect operations on part B, only the breakdown of Cyber Mill 1 is concerned. As the same procedure as Part A, if Cyber Mill 1 is down, Part B coming from dispatcher 1 will go through index table 2 to line 2 and continues its operation on Cyber Mill 2. When Cyber Lathe 2 is down, no effect on the flow of Part B. Figure 3.6 shows the conditional process flow of part B coming from dispatcher 1. The flow of Part B coming from dispenser 2 is the same as that in the production lines.



No.	Operation Description	Machine	Proc. Time (min.)
1	Width Measuring	Width Gauge	0.1
2	Height Measuring	Height Gauge	0.1
3 If Mill 1 on	Face and straight turning	Cyber Lathe	N(6,0.1)
4	Rough and finish milling	Cyber Mill	N(8,0.1)
5 If Mill 1 down	Move to Index Table	Index Table 1	0.14
6	Face and straight turning	Cyber Lathe	N(6,0.1)
7	Rough and finish milling	Cyber Mill	N(8,0.1)

Figure 3.5 Conditional process flow of part A from Dispatcher 1



No.	Operation Description	Machine	Proc. Time (min.)
1	Width Measuring	Width Gauge	0.1
2	Height Measuring	Height Gauge	0.1
3 Lathe 2 on	Rough and finish milling	Cyber Mill	N(4,0.1)
4 Lathe 2 down	Move to Index Table 2	Index Table 2	0.14
5	Rough and finish milling	Cyber Mill	N(4,0.1)

Figure 3.6 Conditional process flow of part B from Dispatcher 1

Figure 3.7 shows the overall path networks for both part A and part B together in the simulation model (represented by the arrows). The difference of this part flow network from the one in production lines is that there are part flows on the index tables. Those flows are optional depending on the running status of the machines. These options are the key points where the extended flexibility of the manufacturing cell lies. They are the advantages that manufacturing cells have over the production lines as well. Figure 3.8 shows a snapshot during the execution of the simulation for a manufacturing cell. The picture illustrates that when Cyber Mill 1 is down, parts are routed from line 1 to line 2, and the robot on line 1 still tries to pick up another waiting part. The behavior of a robot is totally different for the same situation under the production-lines operating settings.

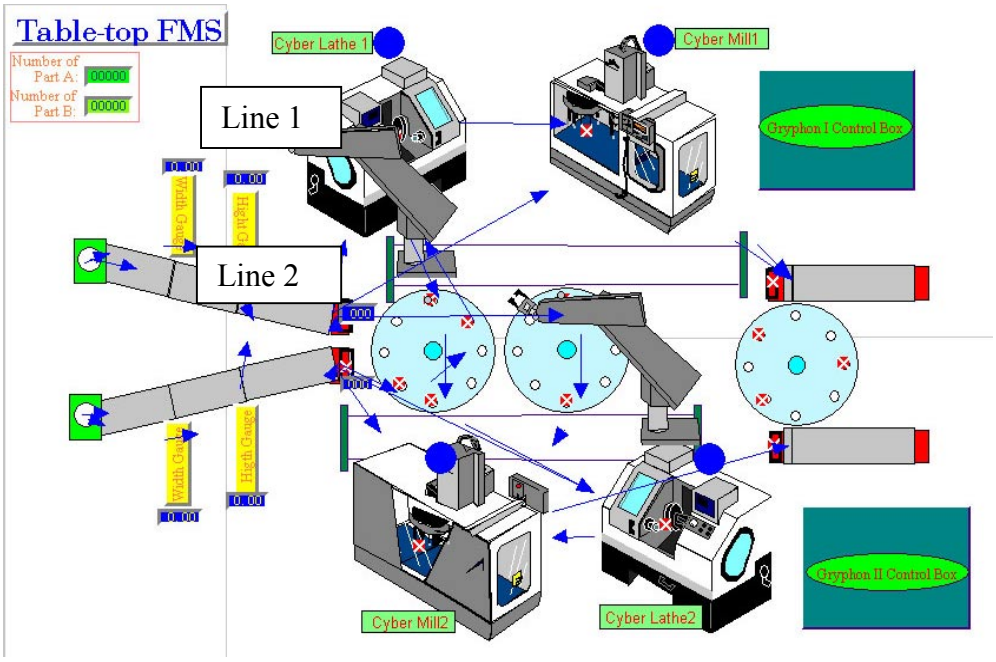


Figure 3.7 Part flow networks in manufacturing cell

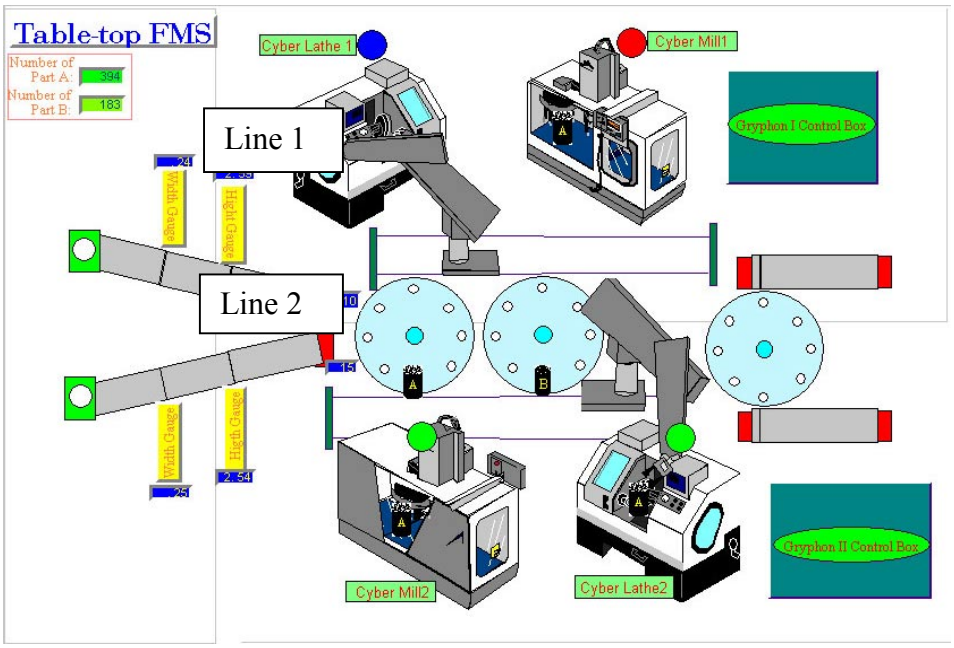


Figure 3.8 Cyber Mill 1 breaks down and parts are routed to Line 2

In the model, all locations compete for the only material handling resource, the robot. It is relatively difficult to estimate how long a part will stay in the FMS because there is no simple calculation for the delay involved in waiting for the robot. But it can be expected that the utilization of machines are relatively low due to the fact that the total expected transportation time (for part A ≥ 4 min., and part B ≥ 2.6 min) is rather significant compared to the total expected processing time for part A and B (for part A is 14 min., and part B is 4 min.), respectively.

3.3.3 Simulation Results and Comparisons

Both simulation models ran for 10 replications. Each replication is composed of a 16-hour warm-up period and a 240-hour effective production period. With animation on, the total execution time of simulation on a Pentium III 450MHz PC is 216 minutes for the model of parallel production lines, and 157 minutes for the manufacturing cell. If running the models without animation, the total execution time is around 66.1 seconds for the parallel production lines, and 72.6 seconds for the manufacturing cell, which are much faster than those with animation.

By animating the simulation, the states of the workstations (idle, busy, blocked or down) are clearly shown at any moment. In addition, accumulative quantities of finished or dumped parts are also clearly shown on the screen. The most impressive scene is that robots really move on the computer screen, nicely mimicking their movements in the real FMS. For example, movements such as picking up the parts from conveyors, moving with the parts held in its gripper, loading the parts into machines, and putting the parts onto the index tables, are carefully positioned and programmed to make them look real. The key to animate the movement of a robot is to capture the continuous statures of robot's movement in the real world, and show them sequentially in a short time.

After the simulation, statistical output reports for resources, locations, entities, and variables can be automatically generated in text and graphic forms. Figure 3.9 is the bar chart to compare the utilizations of all equipments for both models. Figure 3.10 shows the plots of everyday throughput rate of part A and B for the model of the manufacturing cell. Table 3.1 is the summary of the average results of 10 replications for both simulation models.

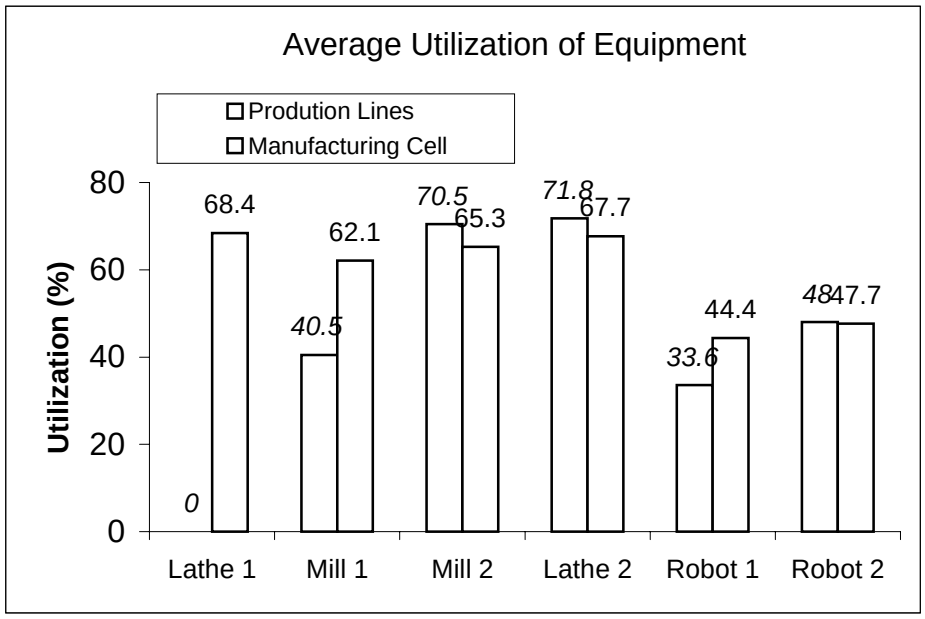
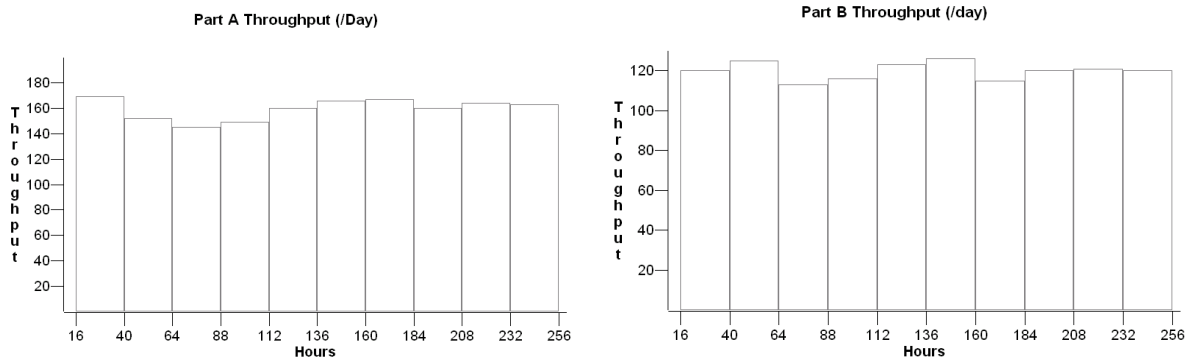


Figure 3.9 Comparison of utilization of equipment for both models



(a) Throughput of part A (b) Throughput of part B

Figure 3.10 Time series of part A and B's throughput for manufacturing cell model

Table 3.1 Simulation results of both models (average of 10 replications)

Interested Parameters		**Production Line**	**Manufacturing Cell**
Execution Time	Animation On (min.)	216	157
	Animation Off (second)	66.1	72.6
Total Throughput (throughput/day)	Part A	1199 (120)	1590 (159)
	Part B	1397(140)	1193(120)
Resource Utilization (%)	Lathe 1	0	68.43
	Mill 1	40.54	62.13
	Mill 2	70.47	65.35
	Lathe 2	71.80	67.68
	Robot 1	33.59	44.42
	Robot 2	47.99	47.70
Average Stays in the System (min.)	Part A	41.21	38.58
	Part B	11.86	20.90
Percent of Blocked (%)	Part A	20.96	25.22
	Part B	3.41	28.62

One might be surprised to find out the total throughput rate of the two models has no significant difference (260/day for production lines, and 279/day for the manufacturing cell, only a 7.7% increase), considering there is one more machine used in the manufacturing cell. However a further calculation of the total effective machine hours reveals that there is a 20.8% increase (373 for the production lines, and 450.5 for the manufacturing cell). As far as the machine utilizations are concerned, the manufacturing cell obtains a more balanced usage among workstations than those in the production lines. The average utilization rate of workstations in the manufacturing cell is 65.9%, with a standard deviation of 0.028. The average utilization rate of the parallel production lines is 60.9%, with a much bigger standard deviation of 17.7. For the parallel production lines, the 95% confidence interval of the mean utilization is ±0.440, and ±0.045 for the manufacturing cell. It is also reasonable that the percentage of the blocked parts in the manufacturing cell is larger than that in the parallel production lines, mainly because of more complex routes in the cell than those for the single flow line. This fact holds especially for Part B. Although Part B requires a shorter processing time, it has to wait longer in the cell because of unavailability of resources. Comparison of these data demonstrates the advantages of using the manufacturing cell over production flow lines. But the ability of complex routing enables the manufacturing cell to handle more types of parts within the cell at the cost of a higher investment on equipment.

To test the effect of random number generation upon the overall computational time, an experiment was designed based on the simulation for the manufacturing cell. The basic idea is that by setting a portion of the random variables used in the model to some fixed values, the total number of random numbers generated for these variables during the simulation can be controlled. For example, there are 20 random variables in the model, which require approximately 255,000 random numbers to be produced for the simulation. If 8 random variables are fixed to their means, the total random numbers needed to produce are reduced to 140,000. A continuous effort can be employed to fix the rest of the random variables until no random variables are used in the simulation. By comparing the execution time of these random variable-reduced models, one can have a brief picture about how large the effect is, with the assumption that the computation time needed to produce one random number for different distributions are almost the same. Figure 3.11 records the results for this experiment. These results show the computational burden to generate random numbers occupies approximately 8.7% of the overall

simulation time. Although it is not very significant in this case, as the portion of the code used to generate random numbers becomes relatively larger because of the simplification of other components of the simulation model, the computational burden to generate the random numbers can significantly affect the speed of simulation.

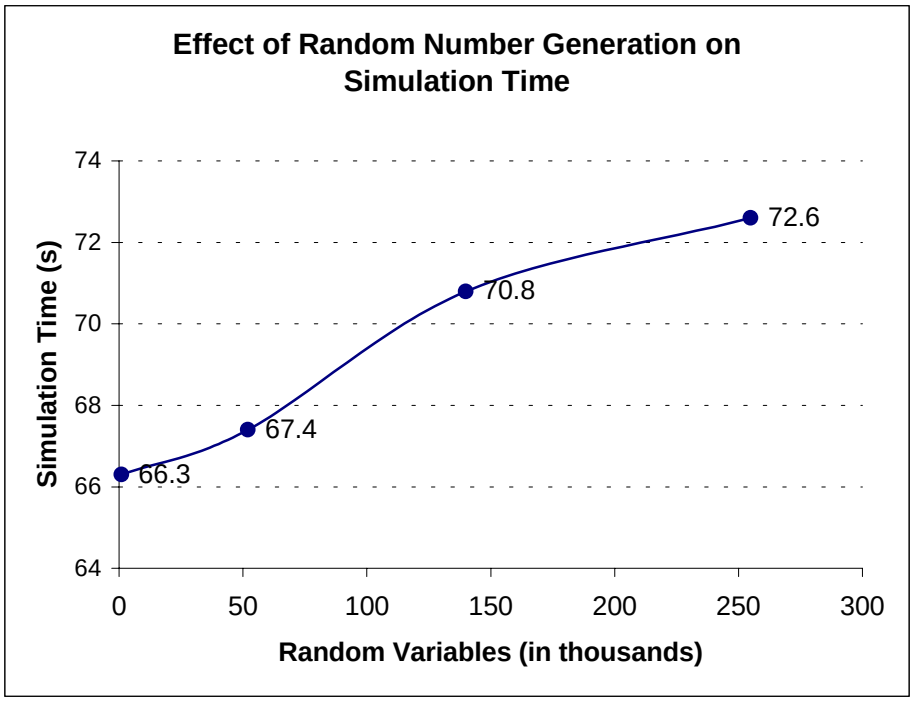


Figure 3.11 Effect of random number generation on total simulation time

Several observations about the factors that can greatly affect the speed of simulation can be drawn from above software-based simulation study. First, the animation, although very attractive, does dramatically slow down the simulation in the order of hundreds of times. That is the reason why nearly all the system simulation packages provide the option to disable the animation. Second, simplification of the control logic can largely reduce the simulation time. According to the statistical nature of simulation, a subroutine in the program may be called thousands of times during a simulation to ensure a statistically sound result. Therefore, a little simplification of the implementation of the control logic can result in considerable time saving for the whole simulation. Under such a situation, the simpler, the better. Extending the same philosophy to the hardware-based parallel simulation, either from the time saving or the

technology standpoint, a simple hardware-based way to represent the complex control logic shall be desirable. Finally, the computational burden to generate tens of thousands of random numbers will become an important issue in the hardware-based parallel simulation. The experiment shows that releasing the burden of generating random numbers has a comparable effect as simplifying the control logic upon the speed of the simulation. Moreover, as the control logic becomes more simplified, the portion of computational time used to generate random numbers can become relatively larger. Even in a software-based simulation of reasonable number of random variables, the computational time spent on generating random numbers is certainly not negligible.

CHAPTER 4

METHODOLOGY AND DESIGN

In this chapter, development of the hardware-based parallel simulation is introduced. Details about the concepts and methodology in support of the hardware-based parallel simulation will be depicted. To help understand these concepts, the laboratory FMS discussed in the previous chapter is employed to serve as an illustration. First of all, because this research seeks to break into a new ground of simulation technology, where even basic concepts are in the development stage, some concepts related to the hardware-based parallel simulation need to be clarified first. Then, available architectures that might lead to the construction of such a hardware simulator are discussed. Those architectures are bus based, shared-memory based, and parallel I/O port based multi-microprocessor structures. Furthermore, enabling digital electronic technologies that can be used to build the simulator, such as DSP, and Multi-port SRAM, are explored. A time scaling simulation method that can be applied for the hardware-based parallel simulation is also developed. In-depth issues relating to the observation of time scaling, synchronization mechanism, and deadlock problems are discussed further. A simple formula is provided to estimate the execution time of such a method. Necessary modifications on the basic time scaling and driven simulation method are also discussed in order to apply it on the multi-microprocessor architecture simulator.

4.1 Introduction to the Hardware-based Parallel Simulation

When hardware-based parallel simulation is referred in this research, it means the execution of a simulation run on a proposed hardware-based simulator for a specific FMS. Hence, the research task can be divided into two main areas: one is to build the proposed simulator and the other is to develop a suitable simulation mechanism to execute the simulation on such a simulator. The development of a simulation mechanism is highly related to what kind of simulator is to be developed. The simulation mechanism can only be tested after the simulator is built.

4.1.1 General Idea of the Hardware-based Simulator

The core of the hardware-based simulator is a multi-microprocessor-based digital circuit board. It is composed of a collection of "micro emulator units" to mimic the complete set of machinery and equipment of an FMS such as machining centers, transporters (conveyor, robots and AGVs), and load/unload stations. The micro emulator unit consists of a microprocessor with its support peripheral chips for local memory and communication. The operating logic of the designated resource, such as processing times for different parts and expected time for breakdown and repair cycle, will be captured via micro-programs for each micro emulator unit. In this way, a printed circuit board (PCB) can be built to represent the total operating conditions of an FMS. The completed PCB can be considered to be a full replicate of a real flexible manufacturing system.

The simulator will run side by side with its designated real FMS. The role of the simulator is illustrated in Figure 4.1. Basically, running conditions of the FMS can be obtained through the I/O ports from the sensors and communications with the cell controller. Thus, all

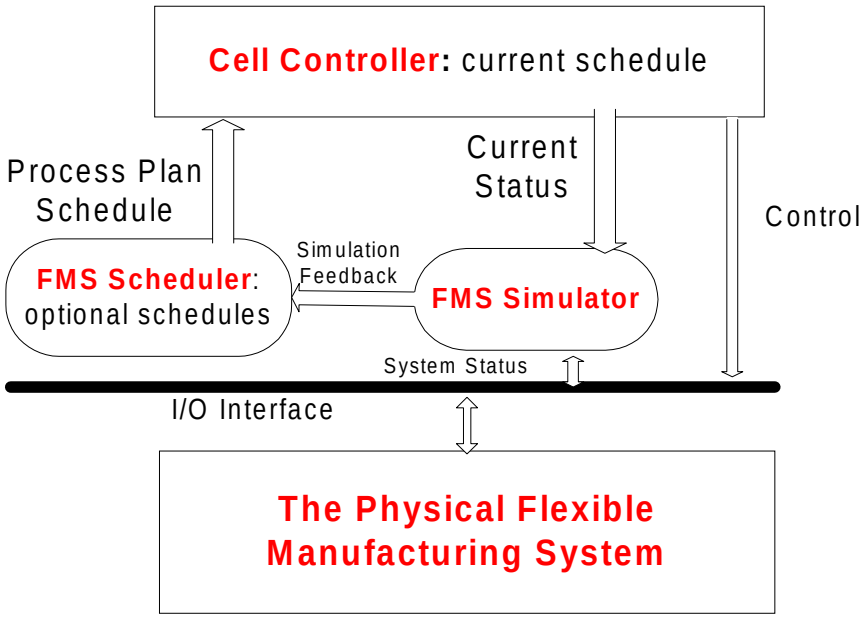


Figure 4.1 Role of the simulator in an FMS

system characteristics such as multiple parts, multiple routings and dispatching rules, multiple processing times, buffer/transporter assignments, etc., are to be fully captured by the corresponding micro emulator units. Then the simulator rehearses the behaviors of the FMS on such a digital circuit board. Simulated performance measures such as machine utilization, total system throughput, work-in-progress inventory, and part flow time can be generated in real-time. The results can then be fed into the FMS scheduler to help it evaluate different schedules or perform rescheduling.

4.1.2 Functions of Hardware-based Simulator

The main function of the proposed simulator is to perform on-line rapid simulation. The simulator will fully emulate the real FMS, which means it would be the simulation model of the FMS. The simulator may work either in a stand-by mode or in an on-line simulation mode. When working in the stand-by mode, the simulator continuously collects current status of all operating machines through sensors input or from cell controller, and keeps updating its records of the operational conditions about the target FMS. When unexpected events occur and the need to reevaluate a new schedule is triggered, the simulator shifts to the on-line simulation mode and performs a rapid simulation under this circumstance. To meet such a requirement, the simulator needs to have following abilities:

- Off-line rapid simulation. Given a fixed schedule for over 20 different types of parts in a medium to large size FMS, the simulator can generate results of an 8-hour simulated work shift within minutes, off-line.
- On-line rapid simulation. When a breakdown occurs, by gathering the current status of the FMS through sensors, the simulator is able to generate the simulation results within seconds, based on the given modified schedule. The new schedule may be a simple modification from the previous schedule, such as transferring the parts originally directed to a down machine to an alternative operating machine, as a remedy for the normal schedule under this special circumstance.

Besides the rapid simulation, with the help of a suitable status monitoring and control mechanism, the hardware-based simulator can achieve the bidirectional simulation for the manufacturing systems. The main theme of bidirectional simulation is a desired coupling of a

model (the virtual factory) directly to a real factory demanding that changes in the real factory be quickly incorporated into the model, or, conversely, that the model be used to drive changes in the real factory [112]. For example, the movement of a part in a simulation model would cause a robot to actually move the corresponding real part in the real FMS. Conversely, the part movement status inside the real FMS can be captured by the simulation model and used to adjust the parameters of the simulation model. The capability of real-time dynamic response to the changes on the shop floor is the basic requirement for bidirectional simulation. Moreover, to enable the tightly coupling, communication links to the real FMS from the simulation model need to be very robust and truly bidirectional.

With the help of an enhanced I/O module and control mechanism, the proposed simulator will be able to achieve the bidirectional simulation. Here, each micro emulator unit must have a direct I/O link with its mapping equipment and communication with the controller of the equipment. As far as data collecting is concerned, the function of an I/O module is to collect the running status through various kinds of sensors on-line so as to fully capture the characteristics of the equipment. For example, power sensing, acoustic-emission sensing or force-monitoring can be used to check the failure of tools. Through the I/O port, the simulator can also automatically record the cycle time for each machine breakdown. Then by some statistical analysis methods, the simulator is able to capture the predicted pattern of tool breakdown, which will help to build a more accurate simulation model. In the other direction, with the help of communication between each micro emulator unit and the cell controller of the FMS, the simulation results can be rehearsed on the real FMS. Depending on simulation results received from the simulator, the cell controller of the FMS can check the real impact on the FMS by running it step-by-step. During this hardware-in-the-loop simulation, each step of state advancing can be checked and status changes are collected. Thus, the simulation model can be refined according to information gathered. The modified simulation results can be transferred to the cell controller again and step-by-step debugging of this hardware-in-the-loop simulation repeats. One thing is clear that the simulator will not take the control from the cell controller, but rather control the FMS through the help of the cell controller.

4.2 Construction Issues of the Simulator

Before building the simulator, several assumptions about the manufacturing system must be in place. First, the layout of the FMS is known. The layout of the FMS includes the number and types of machining centers and their relative positions on the shop floor, the number and types of material handling equipments and their possible transportation path network, the number and types of storage areas and their locations in the FMS. These pieces of information are essential in design such that the FMS can be fully mapped to the simulator. Other details that have no effect on the performance of the FMS, such as power supply or waste discarding equipments, can be ignored. Second, the production requirement for the simulation period is known. A production requirement includes the types of parts to be produced, the lot size for each part, the starting time and due date for each part batch, the required operations for all types of parts, and predetermined part dispatching rules. Basically, a hardware-based parallel simulation scenario is executed to test the performance of fulfilling a production requirement. The information from the requirement is essential to initialize and execute the hardware-based parallel simulation.

To really build a general-purpose simulator from scratch requires a lot of effort and time that can extend beyond the scope of this research. Thus the main effort of this research is to focus on basic theoretical aspects and core technical foundations, which are critical to construct the FMS specific simulator. These foundations include the mapping technology to build the simulator, the system modules of the simulator, and three potential multi-microprocessor architectures of the simulator. These issues are addressed in following sections.

4.2.1 Mapping the FMS to the Simulator

The key methodology for building the hardware-based simulator is mapping (see Figure 4.2). Mapping is carried out on two levels, the upper structural level and the lower equipment level. Structural level mapping refers that the whole structure of the simulator is determined by the layout of the FMS to be simulated. The relative positions of machinery and equipment in the real FMS decide the arrangement of micro emulator units on the printed circuit board. Similarly, the material flow path decides the pattern of data bus that connects microprocessors. As a whole,

the simulator shall look like a lithograph of the FMS on the printed circuit board, although there will be some slight differences for the sake of implementation considerations.

Mapping at the individual equipment level is based on either one-to-one reflecting or multiple-to-one relations. For example, each machining center can have its own reflective micro emulator unit in the simulator while the transportation control center composed of the robots, conveyors, and index tables can be represented by one micro emulator unit.

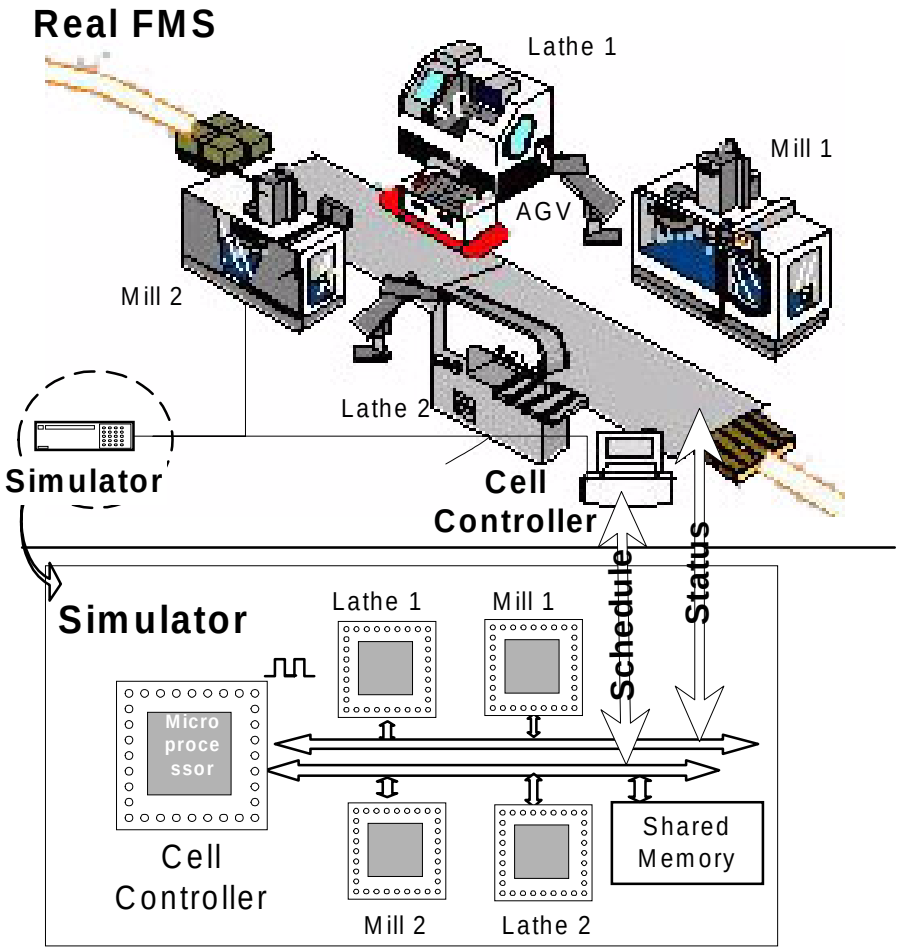


Figure 4.2 Mapping from the FMS to the simulator

Basic components for a micro emulator unit are very similar. They all need some sort of FIFO for I/O operations, SRAM for local memory, E^2PROM for parameter setup, and communication supporting chips (Multi-port SRAM or Bus interface) for inter-processor communications. But depending on the subject to be emulated, one micro emulator unit may have different architecture inside. For instance, a micro emulator unit mimicking the machining center needs some special peripheral chips to simulate the input/output queues, while the other micro emulator unit mimicking the load station needs a digital signal generator to generate different digital signals that represent arrivals of different part types into the FMS. No matter how different it is inside, standard modules for each type of machinery can be designed for easy expansion and integration of the simulator.

To further extend the simulation modeling on the simulator, mapping can be carried out in either a physical or logical way. Mapping the FMS to the simulator corresponds to the modeling methods used to build software-based simulation models. Because of this unique way of building the hardware-based simulator, the method used to build simulation models on the hardware simulator is different from the traditional methods as well. Although the simulator is built by directly mimicking the FMS, simulation models represented by the simulator board can still be built in either physical mapping or logical mapping. The physical mapping refers to above one-to-one mapping method to build the board. Under such a mapping effort, a workstation is emulated by a microprocessor, and transportation equipments (e.g. robots and AGVs) are also emulated by a microprocessor. Parts are emulated by message passing among the microprocessors following communication links that match actual routes that the part will go through the FMS. Overall, physical mapping makes the simulator board become one specific simulation model. Besides physical mapping, simulation models can also be built on the simulator by logical mapping. In logical mapping, several machines that can perform almost the same kind of operations can be mimicked by one microprocessor. This multiple-to-one logical mapping works in such a way that the FMS is divided into submodels for the purpose of parallel simulation. In this case, although the structure of the simulator board no longer reflects the real layout of the FMS, but if it is viewed from the group-level, this structure still captures the essence of the real layout of machine groups. In other words, the simulator board still looks like a lithograph of the FMS but is composed by grouped manufacturing cells. The messages among microprocessors now represent activities between these machining cells and they will be

transmitted to individual machine within the group. Here, inside each microprocessor, there is a traditional discrete-event simulation engine in addition to the proper mechanism to distribute incoming and outgoing messages. The logical mapping methodology is different from the general parallel simulation methods in that the simulator board itself is still the basic simulation model to be executed. The layout of the board is still the layout of the FMS at grouped format. Logical mapping provides some extension of flexibility in modeling the FMS on the simulator board. Moreover, it provides a limited capability for the simulator to simulate different FMS.

4.2.2 System Modules of the Simulator

The proposed simulator shall consist of five functional modules: the input module, the communication module, the interface module, the display module, and the main board (see Figure 4.3) in order to really apply on shop floors. Among them, the main board is the heart of the whole simulator, which contains the hardware-based parallel simulation engine and circuit control. Major functions of each of these modules are introduced as follows:

1. Input Module

The input module is responsible for collecting running status from the FMS, such as states of machines (busy, idle, down, or blocked). The states are stored in a shared States Monitoring Table, which is frequently checked by the central microprocessor to decide the behavior of the simulator. A changing status will trigger the simulator to do the on-line simulation. Within the module, in addition to the digital sensor inputs, there may be some Analog-to-Digital inputs to monitor continuous status. The heart of this input module is a microprocessor to provide the simple ability of confusing signals from several monitors to identify a problem and maintain the States Monitoring Table.

2. Communication Module

The function of the communication module is to provide the communication between the simulator and cell controller. Through the communication module, the simulator obtains the information about the current schedule and passes simulation results to the cell controller. A cell controller or scheduler may use this information to further reschedule and control the FMS.

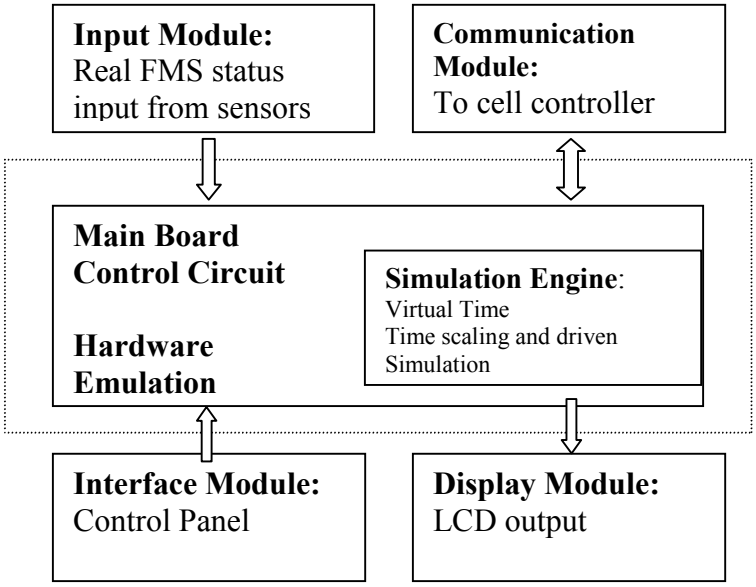


Figure 4.3 Functional modules for the proposed simulator

3. Interface Module

The interface module provides a human interface to the simulator. Through the control panel, the operator can set up the operating mode of the simulator, parameters for the simulator, etc.

4. Display Module

The display module displays the running state of the simulator and the simulation results on an LCD display. It is also possible to display animation of one simulation run on the board when the simulator goes off-line, providing a visual tool to verify the simulation.

5. Main Board

The main board is the heart of the whole system and the main focus of this research. It performs the core hardware-based parallel simulation and controls other modules. The term "simulator" used in this work typically refers to this main board. Basic issues addressed in this research are what kind of multi-microprocessor architecture is most suitable for such a board.

4.2.3 Investigation on Possible Architectures for the Simulator

The very first step to construct the simulator is to decide the architecture of the simulator. Three multi-microprocessor architectures that might be suitable for the main board of the simulator are briefly investigated. These architectures are classified by ways of communication between microprocessors. They are, namely, bus-based, shared-memory-based and parallel I/O port based systems.

The bus-based multi-microprocessor architecture is shown in Figure 4.4. The architecture makes use of some commercial buses suitable for parallel processing, such as VMEbus, to build up the multi-microprocessor structure. The microprocessors work in a master/slave relationship. The I/O and communication between microprocessors are controlled by the master microprocessor, which is also the one emulating the functions of the cell controller in a real FMS. According to instructions from the master microprocessor, slave microprocessors can get I/O and data through bus. Moreover, if the slaves want to communicate with each other, they must first send requests to the master to obtain the permission from the master.

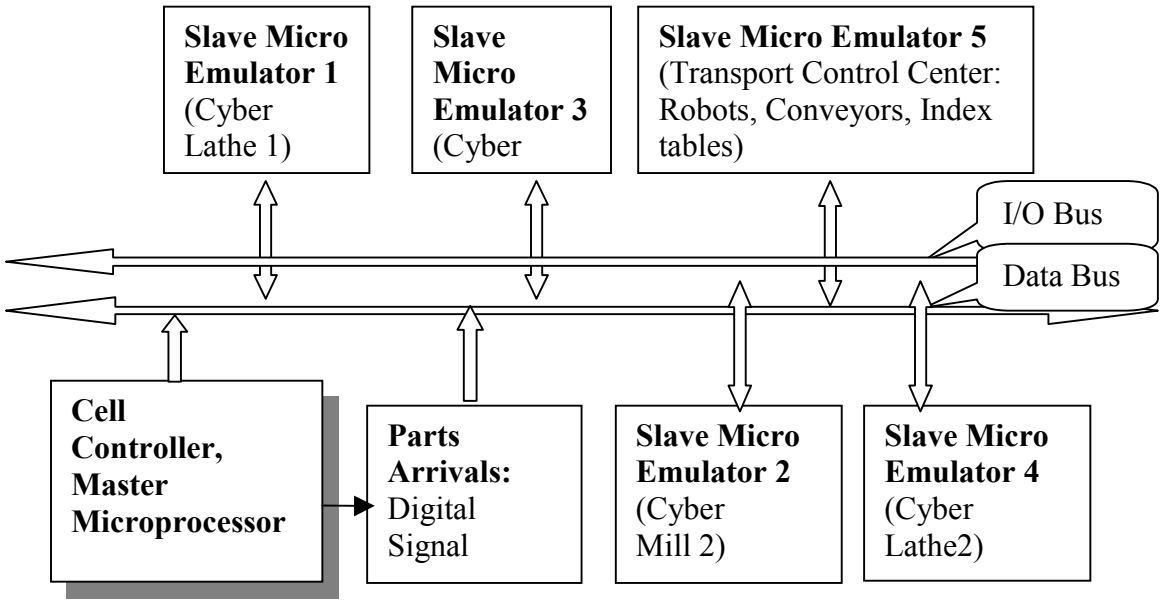


Figure 4.4 Bus-based multiprocessor architecture for the simulator

The advantage of this structure is that the Bus technology is standard and very mature, so it is relatively easy to obtain technical support and to expand the system later on. Most of the time, the expansion tasks are as simple as plugging the new board into the slot, then followed by performing some necessary settings. But the disadvantage is also very obvious. With all the communications controlled by the master microprocessor, communication ports on the master microprocessor will be the bottleneck of the whole system. The communication delay caused by congestion on the data bus may significantly slow down the speed of the simulation.

The shared-memory-based multi-microprocessor architecture can solve above problems (see Figure 4.5). Here each microprocessor has its own local memory to store local information, such as parts routing instructions, processing times for different types of parts, time series data recorded during the simulation, etc. The simulator has a shared memory which is accessable to all microprocessors and stores common information, such as messages between microprocessors, machine status, etc. A bus arbitrator is added to take care of possible collisions on the data bus.

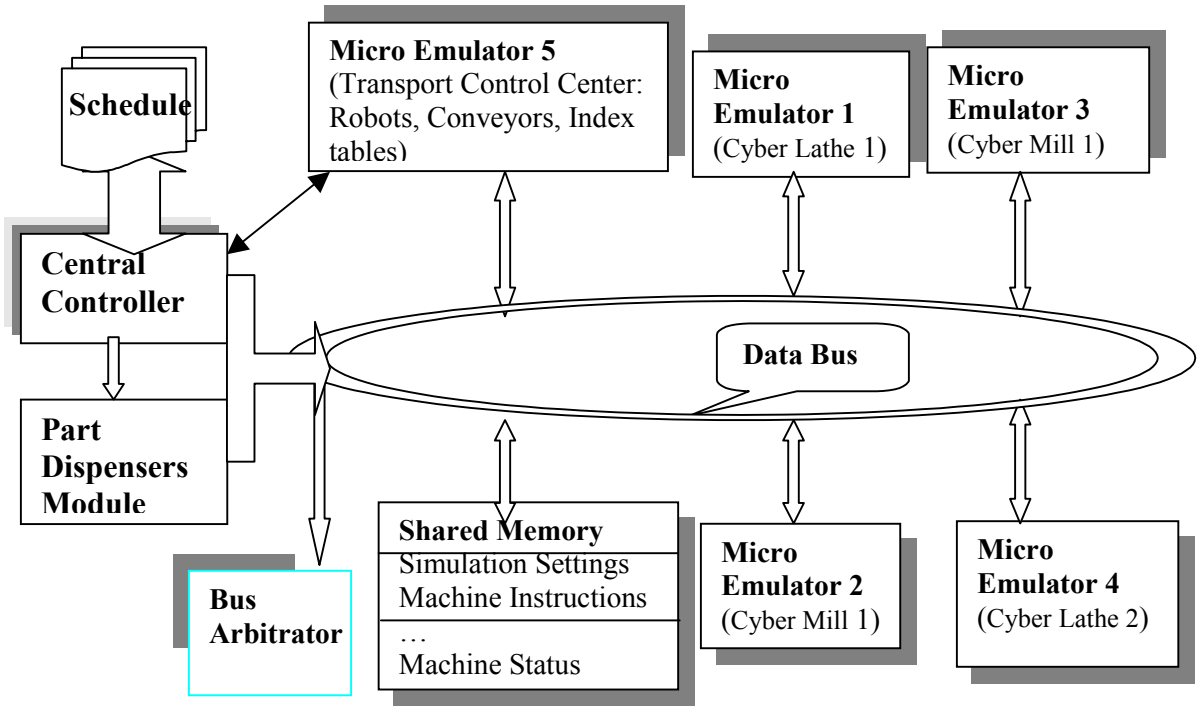


Figure 4.5 Shared-memory-based multiprocessor architecture for the simulator

Compared to the purely bus-based system, the master microprocessor has more time to perform central control functions because it need not be in charge of all communications. The working procedure of the simulator may act as follows: the master microprocessor first generates the simulation settings according to the schedule, and stores them in the shared memory. Then microprocessors emulating machines and the transportation subsystem will fetch individual settings for them to their local memory. Once the simulation begins, the master microprocessor acts as the central controller to control the advancement of the simulation. The master microprocessor can even stop the simulation if needed at any midpoint during the simulation. Other microprocessors will follow those simulation instructions that they receive and update their status in the corresponding storage space in the shared memory for the synchronization of part flow, while at the same time record the progress data for the time series varaibles in the model and then store them in their own local memory. In this way, communication within the system is relatively faster that the bus-based architecture, but there does exist a coherency problem to synchronize the shared-memory. The complex bus arbitrator method plus the coherence mechanism for the shared-memory are very difficult to realize.

Among the three architectures, the parallel I/O port based architecture is the simplest and the easiest one to realize. The framework of this architecture is shown in Figure 4.6. Basically, the communication within the system is provided by point-to-point links through parallel I/O ports. The information is exchanged by passing messages. For example, during the simulation, after one robot in the transportation system finished the operation of moving the waiting part from the conveyor to the lathe, the transportation system emulating microprocessor will send a message to corresponding microprocessor to indicate that the part is loaded on the lathe. Then the machine tool emulating microprocessor will begin to simulate the machining operation and set new status of the lathe and the part. Obviously, this architecture requires that the microprocessor possesses several parallel ports so that it can be directly linked to all other microprocessors on the board. If there are numerous machines in the FMS, this architecture will be infeasible because of its limited on-chip parallel ports. One solution is to use a dual-port SRAM to link two microprocessors directly, instead of using the on-chip parallel ports. Usually the synchronization of the dual-port SRAM is realized in hardware, so the communication capacity can be expected to be almost the same as parallel ports.

There are two possible electronic technologies to implement individual micro emulator, one is to use a simulation customized IC chip, and the other is to use microprocessors designed for parallel computing. Using a simulation customized IC chip can achieve fast speed but at the cost of losing some flexibility of the simulator. Another problem is that it required other research effort to build such a chip first because currently such an IC chip is not yet available. Because it is a customized design integrated circuit chip, the cost could be an important concern. On the other hand, microprocessor technology is very matured and can provide some sort of flexibility to the simulator, but at the expense of losing some speedup. Therefore, to choose which way to implement individual micro emulator is really a compromise between the flexibility of the simulator and the speed of the simulation execution as well as the cost issue.

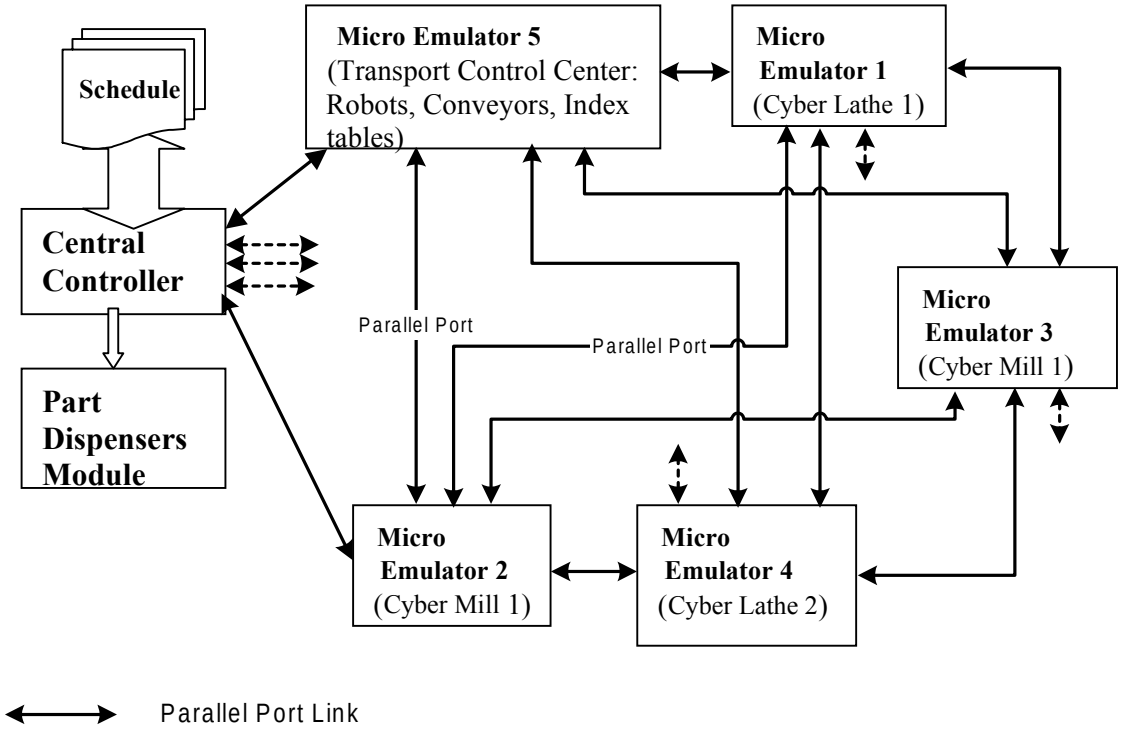


Figure 4.6 Parallel I/O port based architecture for the simulator

For the simulation customized IC chip design, the idea is to use a digital circuit composed by electronic logic gates to realize the simulation logic and advance the simulation execution. Basically, such an IC chip consists of an input data receiver, an inside server logic control, and an output controller, as shown in Figure 4.7. A simulation engine can be viewed as a composition of routines and logic control to process simulation events. All those routines and logic control can be implemented by some kind of AND/OR logic gates. For example, at the input side, part arrival can be represented by electrical pulse, and the digital cycles between these pulses stand for the time between arrivals. The part type can be decoded from other parallel I/O ports at the same time with the pulse. The electrical pulse and part type information can be sent from an outside signal generator, which means a part arrival from the loading stations, or it can be sent from other such chips, which means a working-in-process part arrival. The status of other servers, such as machine broken and WIP queue full, can also be obtained by a direct wiring to that server. The data receiver receives all these input signals, decodes them, and feeds then into the input logic control unit. The input logic control unit performs the same function as subroutines in conventional simulation methods to process arrival events. Basically, the logic to process an arrival event can be described as follows. If the workstation is busy, put the arrival part into the queue; else, let the workstation process the part and set the workstation to be busy.

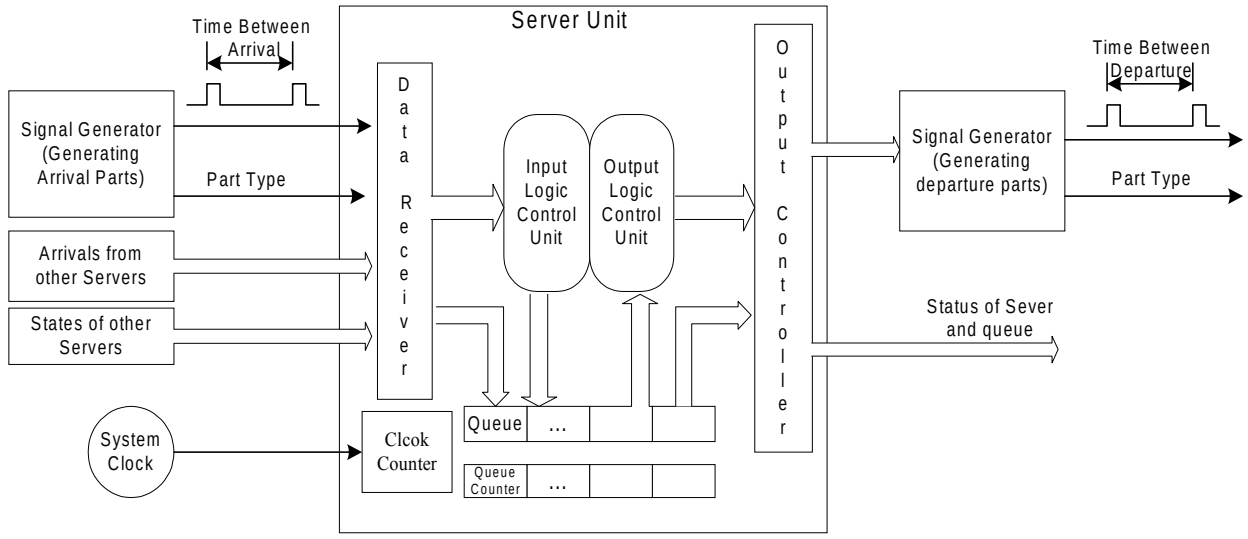


Figure 4.7 Design of a simulation customized IC chip

In the input logic control unit, such logic is implemented by AND/OR electronic gates, which takes signals indicating part arrival and current machine status, and generates a 0-1 control signal to move the arrival parts information either to the output logic control unit directly (if 0, machine idle) or to the queue (if 1, machine busy). Likewise, the output logic control unit controls another signal generator, which can generate electronic signals to other chips, indicating a part departure from this workstation. The status of this workstation is also sent to other chips through the output logic control unit. All these operations are performed concurrently driven by a system clock. The advantage of this implementation is that the simulator is realized in true electronic components and may achieve very high speedup to the simulation. The disadvantage of the method lies in the difficulty to realize more complex logic in electronic gates and lack of flexibility to accommodate possible changes of the simulation model. Therefore, it may only be suitable for simple structure manufacturing systems simulation.

The research for the above customized simulation chip will be carried on by other researchers in a contingent phase of this project by using FPGA technology. In this research, however, microprocessors are the main computing power for individual emulators. By using microprocessor, plus the same supporting peripheral circuits, certain extent of the flexibility for the simulator can be obtained at the cost of losing the possible high speedup of customized simulation chips. In microprocessors, the routines and logic control in the simulation are realized by assembly language programs. The micro-programs can be easily reprogrammed to reflect the dynamic changing within the physical FMS.

Following describes one possible simulation run on such a simulator. On the simulator, each microprocessor has its own local memory and communication links to all other microprocessors. During the initialization phase, each micro emulator will read its local simulation parameters, such as the processing time for different kinds of parts, input/output queue length, the pattern of MTBF and MTTR, etc., from the programmable E^2PROM. The E^2PROM can be easily reprogrammed if any of those parameters changed according to the real-time status of its corresponding machining center. The master micro emulator, mimicking the cell controller, will generate settings realizing the dispatching rules, and pass them to other micro emulators. These settings contain the information about which dispatching rule to be used by individual micro emulator so that it knows from which upper stream micro emulator it can expect to receive in-coming messages, and to which downstream micro emulator it should send

the out-going messages. Once all micro emulator units setup the simulation parameters, the simulation begins. During the simulation, the master microprocessor will control the digital signal generator to generate the part entry according to the part types and arrival patterns. Each micro emulator unit will keep on receiving and passing the digital signal following the route that a part travels within a real FMS. If necessary, the micro emulator unit can update its status stored in a reserved space in the shared memory for other micro emulators to reference. For example, it will update its WIP queue full flag to the upper stream micro emulator so that these upper stream micro emulators can use this flag in the logic control to see whether they are blocked during the simulation. During the simulation, each micro emulator records its own time series data, such as the number of finished parts, waiting time for each kind of part in a queue, blocked time for the machine, period of machine broken and repaired, etc. These records are stored for the final statistical analysis after the execution. When the simulation ends, each microprocessor will first collect its own record and perform the statistical analysis in parallel. Then necessary information will pass among microprocessors to generate the final simulation results.

A brief comparison of these three architectures in technical innovation, realization risk, cost, and available technical support is summarized in Table 4.1. Which architecture to choose really depends on the specific FMS to be simulated, total developing cost and available developing time. Realistically, a combination of the architectures can be a good choice to build the simulator.

Table 4.1 Comparison of the three architectures

Features	**Bus-based**	**Shared Memory-based**	**Parallel I/O port-based**
Technical innovation	Medium	High	Low
Realization Risk	Medium	High	Low
Developing Cost	High	Medium	Medium
Available Technical Support	Mature commercial market	Only microprocessor support	Appear in literatures

4.3 Enabling DSP Technologies for the Simulator

One of the main technical challenges of this research is to find out suitable microprocessors for building the hardware-based simulator. Some important requirements used as the guidelines to select the microprocessor are:

- Capability of performing parallel processing and multiprocessor communication.
- Support of multiprocessor programming, compiling and debugging.
- High computing power.
- Relatively low cost.

Recently, the advancement of digital signal processing technology attracts immense attentions from researchers. Initially, digital signal processors (DSP) are specifically designed for pure digital signal processing applications predominantly in the fields requiring mass computation, such as image processing and computer voice analysis. Nowadays, their applications are extended to control and parallel simulation because of their powerful computing ability (32bit float point) and inherent hardware designs for parallel processing [59],[60]. Among the hundreds of types of microprocessors investigated, one digital signal processor TMS320C40, developed by Texas Instrument Inc., can best meet all these criteria.

The TMS320C40 device is a 32-bit floating point Digital Signal Processor optimized for parallel processing. The C40 chip combines a high performance CPU and DMA controller with up to six communication ports to meet the needs of multiprocessor and I/O intensive applications. It is also compatible with TI's multi-chip development environment. Each chip contains an on-chip analysis module which supports hardware breakpoints for parallel-processing development and debugging. The features of the TMS320C40 are listed as follow:

- High-Performance 32-Bit Floating-Point Digital Signal Processor (DSP), TMS320C40-50 (5 V), 40-ns Instruction Cycle Time, 275 MOPS, 50 MFLOPS, 25 MIPS
- Harvard-architecture, with two identical external data and address buses. Bus lock/unlock signals are available to support shared memory systems.
- On-chip parallelism (pipelining, multiple ALUs, multiple buses).

- Six communication ports for direct asynchronous processor-to-processor communication, bidirectional, parallel interfaces with a transfer rate of 20 MBytes/s each.
- Six-channel DMA coprocessor with a special mode to serve the six communication channels. CPU and DMA-channels can work in parallel.
- On-chip JTAG-standard analysis module which support hardware breakpoint for debugging parallel processing.
- Interlock assembly instructions for parallel processing development.

The problem of the TMS320C40 is its on-chip memory is small. External SRAM must be added. It is worth mentioning that there is a unique microprocessor, TMS320C8x (the Multimedia Video Processor also developed by Texas Instrument), integrating as many as four parallel-processing DSPs and a 32bit RISC master processor on one single chip. With this trend of integrating multiple microprocessors on one chip, it is possible to realize a whole hardware-based simulator on one single chip in the future.

Based on the TMS320C40 chip, the TMS320C40 Parallel Processing Development System (C40 PPDS) is the first development board designed exclusively to evaluate and develop parallel-processing, floating-point software applications. It can be used to develop, benchmark, and evaluate parallel programs in real-time with the power and speed of the TMS320C40 PPDS. The C40 PPDS is a stand-alone target system (with power supply) that is placed on the desktop and is controlled through an emulator. The board has 4 TMS320C40 DSPs, 64K words of local SRAM, 8K bytes of EPROM, and 128K words of shared global SRAM. The structure of the C40 PPDS is actually a combination of the parallel port based and shared-memory based architecture. Each TMS320C40 on the PPDS has direct connections to each of the other TMS320C40s in the system through the communication ports, and can access the 128K global SRAM. These features and more give the user the flexibility to distribute tasks between multiple processors and to develop, compile and debug multiprocessing algorithms.

4.4 Modeling Elements on the Simulator

To benefit from the special hardware features of the simulator, a hardware-based parallel simulation method is more suitable than traditional parallel simulation methods. In following sections, the research focus is switched from the hardware design of the simulator to the software programming for the hardware-based parallel simulation. General issues about how to represent components of the FMS in the hardware-based parallel simulation and simulation modeling elements are introduced first. Then a modified time scaling and driven method is proposed as a possible engine to drive the hardware-based simulation on the simulator. In depth issues related to the observation of time scaling, synchronization mechanism, and deadlock problems are discussed further. For the hardware-based method, the simulation speed can be estimated by a simple formula.

4.4.1 Representing FMS Components

To construct the hardware-based parallel simulation mechanism, there are several questions to be answered first. How should components of the FMS be represented in the hardware-based parallel simulation? Based on these representations, how should the simulation be executed on the simulator? What is the way to collect and generate the statistical results? Before developing the hardware-based parallel simulation method, all these aspects should be considered in depth. In this section, how to represent components of the FMS in a way that is suitable to the hardware-based parallel simulation is introduced. Issues relating to the execution procedure of the simulation are then discussed.

In simulation of the manufacturing system, from the component's view, a simulation model may consist of parts, workstations, storages, resources, the routing path network, and the production schedule. These components are the basic elements in software-based, and hardware-based parallel simulation as well. The difference is that they may be represented in different ways.

1. Parts

Parts refer to the items being processed in the FMS, including raw materials, piece parts, assemblies, loads, WIP and finished products. Parts may be assigned with attributes that can be

tested in making decisions or for gathering specialized statistics. The common attributes associated with a part are: part type, required operations, processing time of each operation, starting time and finishing time of each operation, and other elements needed for special decision logic. In the hardware-based parallel simulation, parts are still identified by their types in the proposed time-driven method, the same as the software-based simulation. It is possible, however, in a true hardware-based parallel simulation, to represent the part in different types of digital signals, such as the amplitude level or the length of a digital pulse.

2. Workstations

Workstations are fixed places in the FMS (e.g., machines, preparing station, load/unload station) to which parts are routed for processing. In the hardware-based parallel simulation, machines are mapped to microprocessors. The characteristics of individual machine, such as the capability to perform certain manufacturing processes, the processing time for different types of parts, the statistics on mean time between failure (MTBF) and mean time to repair (MTTR), can be captured by the microprocessor. Different from the software-based simulation, which relies on other hardware to check the running status of machines, the hardware-based simulator, with each microprocessor directly monitoring its machine, can possibly achieve truly on-line simulation and simulation-based control of the FMS.

3. Storages

There are two types of storages in a typical FMS, buffers and the central storage area. Buffers are mainly attached to a machine to hold the incoming or outgoing parts. The central cell storage area is used to store work-in-process (WIP). There is a third type of storage that does not exist in FMS but might exist in a simulation model, the decision points. They are adopted as the place where parts are temporarily stored to make some decisions about its further routing path. The storages are usually modeled as queues. Moreover, these queues are manipulated according to the dispatching rules in the schedule. To accurately simulate the dispatching rules is the key to ensure accurate simulation results from the hardware-based parallel simulation. In the future direction of the simulator, a customized microchip to emulate queues and associated dispatching rules may be possible. This microchip works like a “smart queue”, which is able to sort the elements in the queue in correspondence with the dispatching rules. In this research, because

there is no such microchip available, queues are still simulated in traditional ways, that means they are represented by arrays, lists, and hash tables.

4. Resources

A resource may be a person, tool, vehicle or other object that may be used to transport material between workstations or help to perform an operation on a location. A resource may be either static or assigned to a path network for dynamic movement. There shall have one dedicated microprocessor on the simulator to manage all the resources. Before any attempt to transport finished parts by a workstation, it has to acquire the transportation resource (such as a robot or an AGV) from the resources controller first. In order to obtain necessary transportation resources, a microprocessor sends a request to the resources controller. Upon receiving the request, the resources controller will allocate the resource to the microprocessor by sending a permission token to the microprocessor, if the resource is free at that time. If the resource has been occupied, the resource controller will hold the request until the resource is released by other operations. Then it will send the permission token to the request microprocessor, along with the waiting time for the resource. In the meantime, before getting permission from the resource controller, the microprocessor will hold the transportation operation, and it may perform other operations until the resource is received. In this way, the resources controller is responsible for simulating the competition to resources. In this research, the tool management issues will not be considered.

5. The Path Network

The path network defines all the possible paths that entities and resources may travel when moving through the FMS. The path network consists of nodes and path segments connecting these nodes. Here, nodes are microprocessors, and path segments are communication links among these microprocessors. On the simulator board, each microprocessor has some ways to be linked to other microprocessors, either directly or indirectly. Direct links are point-to-point communication by means of parallel ports or shared-memory, while indirect links means by bus or message passing with the help of intermediate microprocessors. The exact path network through which a part may follow is determined by dispatching routes adopted in the schedule.

6. Routing

Routing defines the processing sequence and flow logic of entities between routing locations, including part input-output relationships such as assembly or disassembly, parts sequences, flow logic, routing conditions, moving times, etc. In a real FMS, routing is realized by decisions from the cell controller on how to move the individual part considering the preset dispatching rules and current system status. These decisions are usually made in real-time and enable the cell controller to control the cooperation of transportation equipment so as to move up the whole FMS and realize the schedule. Therefore, how to simulate the routing features becomes a critical problem for all simulation models, no matter whether it is a software-based or a hardware-based parallel simulation. In software-based simulation, the flow logic is implemented by careful programming logic. It is also true for the hardware-based parallel simulation, but different in the programming language. In the hardware-based parallel simulation, the part movement is represented by digital signals passing between microprocessors. The dispatching rules, such as FCFS (first come first serve), SPT (shortest processing time), SLACK (least slack time to meet the due date), EDD (earliest due date), and other more complex dispatching rules, can be implemented by programming the corresponding logic in assembly language. Furthermore, because of the FMS specific structure of the simulator where each microprocessor controls its own part routing, the realization of dispatching rules can be simplified and executed in parallel on all microprocessors. The scheme allows each microprocessor to decide where to send a part, instead of having a central transportation controller to do so. This divides the complex computational task in realizing the dispatching rules into several simple tasks running on separate microprocessors. In this way, this portion of the computational time can be reduced, depending on the number of microprocessors used.

These components can be used to build the simulation model on the simulator, similar to the modeling elements of software-based simulation used to build the simulation model. After successfully representing the components of the FMS in the hardware-based simulator, the next challenge is to program such a hardware-based parallel simulation on the proposed simulator. It is required to explore inside functioning elements of the hardware-based parallel simulation.

4.4.2 Internal Programming Elements

To program for the hardware-based parallel simulation, some important internal elements related to a simulation procedure must be studied further. These internal elements include states, events, timing method, random number generating method, and statistical collection. Some of these are the same as the software based simulation, some are slightly different, and others are totally different.

1. States

A state describes a constant behavior or characteristics of a component in an FMS for a time period. All machines, parts, resources have their own status during a simulation. The overall system state is a collection of status of machines, parts, and resources at the same time period. From the theoretical aspect, the execution of a simulation is to generate a state sequence. Thus, a key step in carrying out a simulation is to define suitable states of the system. The state must maintain sufficient information so that pertinent quantities can be measured and combined to yield meaningful estimates for major characteristics of the FMS. In the hardware-based parallel simulation, the states of each component are defined as follows:

States of Parts: A part has three states. It can be in processing, waiting or moving during a simulation. These states have no use for the execution of such simulation but help to calculate the statistical results related to parts when simulation is finished.

States of Workstations: A workstation can be either in busy, idle, broken, or blocked state during a simulation. When there is a part being processed on the workstation, the machine is busy, otherwise, it is idle and can receive next part. When a machine breaks down, it is in the broken status and needs repair. A blocking situation can happen when a workstation tries to send the finished part off to its next destination machine that is busy and all available queues are full. Those queues include the outgoing queue for the sending workstation, the incoming queue for the receiving workstation, and the queue for the central storage area. In such a situation, the part has to stay on the machine until a space has opened up for it. Of course, the blocked machine cannot perform any other operations. During a simulation, the states of workstations are stored in a shared memory which is accessible to all the microprocessors. Whenever the state is changed, it will be refreshed in the shared-memory immediately. A workstation can check the status of its

downstream workstations in the path network by reading its status from the shared-memory. If no shared-memory is available, the workstation needs to inform whatever changes of its status to its upstream workstations immediately, and stores such information receiving from its downstream workstation.

States of Storages: A storage area has two states, full or empty. When a queue reaches its capacity, blocking may occur. The information of storage areas is also stored in the shared-memory.

States of resources: Resources are controlled by a resource controller, with states either as occupied or free. Once a resource is assigned to a request, it is marked as occupied and other requests for the same resource can no longer be granted until it is released and becomes free.

Overall, states are useful for the control logic during the simulation, because most of the control logic needs to check these states. States can transfer from one to another when an event occurs.

2. Events

Events are activities that cause the change of states, usually happening in a very short time span that is only considered as a time point in the simulation. The process of a discrete-event simulation on a finite state space makes state transitions in accordance with the occurrence of events associated with the occupied state, and new events are generated and scheduled during state transition. In the hardware-based parallel simulation, three types of events need to be considered. They are, namely, parts arrivals, parts departures, and machine breakdown. Different from the traditional software-based simulation, parts arrivals or departures happen not only at the entrance of the simulation model, but also at every microprocessor mimicking a workstation. Parts arrivals are represented by receiving messages. The occurrence time is the time when the message arrives. In the same way, parts departures are represented by sending messages. Machine breakdowns, however, are the internal messages to a microprocessor that can be realized by means of a trap, a software generated interrupt. Events are stored in the local event list and maintained by individual microprocessor, and no central event list is used in the hardware-based parallel simulation.

3. Timing

Timing refers to maintaining the correct occurrence sequence of events such that the causality principle would not be violated. To record the absolute time when an event happens is not important. The important thing is that the sequence to pick up the event from the event list to advance the simulation clock must be consistent with those sequences happening in the real world. In the traditional simulation, the problem is solved by using an internal clock to sort events. In software-based parallel simulations, because of the distribution of virtual clocks and the arbitrary communication delays, an additional synchronization mechanism is needed to keep the correct occurrence sequence of events. Therefore, a significant portion of computational time is spent on synchronization during the simulation. On the hardware-based simulator, because all microprocessors use the same crystal clock, there is a possibility to reduce the overhead of synchronization by using this feature. One obvious option is to use this crystal clock as the global clock for the conservative parallel simulation, and let each individual microprocessor click its own local clock using its timer. However, with the ability to manipulate the computation time, a time scaling simulation method can benefit most from the special structure of the simulator. Details about this method are introduced in next section.

4. Random Number Generation

In simulation models with random variables, generating exactly desired distributions for these variables is critical to ensure the accuracy of the simulation results. Unfortunately, it takes a considerable portion of valuable computing time to generate these pseudo random numbers when executing a system simulation process. As the control logic is simplified in the hardware-based parallel simulation, this portion of computing time will become relatively larger. For example, in the simulation of a simple queuing network, the time needed to generate the random numbers occupy roughly 23% of the total execution time. Therefore, if one can find a way to reduce the time spent to generate the random numbers, the simulation speed can be dramatically improved. One solution is to search for a faster algorithm to generate random numbers. The other direction is to realize the random number generation by hardware so as to release the computation burden from the main computing power. In this research, the random number generation methods are still implemented through software subroutines.

5. Statistical Data Collection

Statistical data collection is to calculate and report the simulation results. Basically, each microprocessor is responsible for recording the interested features of all activities executed on it. For example, the parameters about a machine, such as the utilization, percentage of blocking, percentage of breakdown, MTBF and MTTR, will be recorded and calculated by a corresponding microprocessor. The overall FMS performance, such as the throughput, tardiness of the work batch, and average time for parts staying in the system, will be calculated by the microprocessor simulating the part entering and leaving the FMS. The parameters associated with the transportation equipment can be generated by a microprocessor designated as the resource controller. If some special features needed information involving several microprocessors, for example, the average waiting time for parts to be processed, the microprocessors will send the information to one dedicated microprocessor to calculate such a feature. In such a parallel computing way, the computational time needed for statistical reporting is reduced.

Among all these aspects, the timing mechanism is the most important, and usually used to distinguish different simulation methods. In software-based parallel simulation, because of the distributed simulation clocks and the communication exchanges, an additional synchronization mechanism is needed to keep the correct occurrence sequence of events. This is true for both Parallel Discrete Event Simulation and hardware-based parallel simulation. Under such a sense, studying the timing mechanism of general parallel simulations will provide a valuable insight to the timing mechanism for hardware-based parallel simulation. Besides the conventional parallel simulation methods, by making use of the unique hardware feature on the simulator, advanced parallel simulation methods can be developed to achieve a faster speed. On the hardware-based simulator, because all the microprocessors use the same digital clock, there is a possibility to reduce the overhead of synchronization by using this feature.

4.5 The Time Scaling Simulation Method

A time scaling simulation method is one possible way to make use of hardware features on the simulator to facilitate the simulation. Some theoretical issues about the method, including the observation of time scaling, the description of its synchronization mechanism, the

justification of its deadlock-free feature, and its potential speedup, are discussed in following sections.

4.5.1 Observation of Time Scaling

Several basic issues must be considered before further details of the time scaling simulation method is developed. These issues form the theoretical base for the method. The most important issue for hardware-based parallel simulation, as well as for PDES, is to maintain the local causality constraint in order to guarantee the simulation results are "correct". The results of the hardware-based parallel simulation are said to be "correct" in the sense that these results are the same as those generated by a sequential simulation for the same simulation scenarios. To ensure the results are correct, the local causality constraint must be satisfied. The local causality constraint is stated as follows:

> A discrete-event simulation, consisting of logical processes that interact exclusively by exchanging time stamped messages obeys the local causality if and only if each LP processes events in non-decreasing time stamp order.

The local causality constraint is sufficient to guarantee that parallel simulation is "correct". Theoretically, every simulation execution can be viewed as a process of generating a sample path in a state space, which is composed by a vector of state variables, as shown in Figure 4.8. The following observation about the simulation time and the computational time forms the foundation for the time scaling simulation method:

> If simulation time ε for one state changing from S(t) to S(t+ε) can be directly scaled down to a short period of computing time λ of the corresponding computation task needed to generate such a status change during the simulation execution, and all such scaling processes for all status changes use the same value of this time scale, then it is sufficient to satisfy the local causality constraint.

To substantiate above observation, two situations need to be considered. One is to consider using one LP to simulate several state variables, and the other is to consider several state variables processed by different LPs. The first case is relatively easy to justify by the example in Figure 4.8. In Figure 4.8, two state variables, 1 and 2, are to be simulated by one LP, LP1. Suppose state variable 1 generates an event A at simulation time T0, and state variable 2

generates an event B at simulation time T1, where T1 > T0; and event A leads to an event C happening later at simulation time T2, where T2 > T1. Thus, in the physical system, these events actually happen in the order of A→B→C. From the observation, it can be calculated that the LP1 processes event A at computation time T0/S, where S is the scaling factor, and processes event B at computation time T1/S, and event C at computation time T2/S. The sequence of LP1 computing is A→B→C, just the same as what actually happens. Therefore, in any circumstance, event C will not be processed before event A and B, which ensures the local causality.

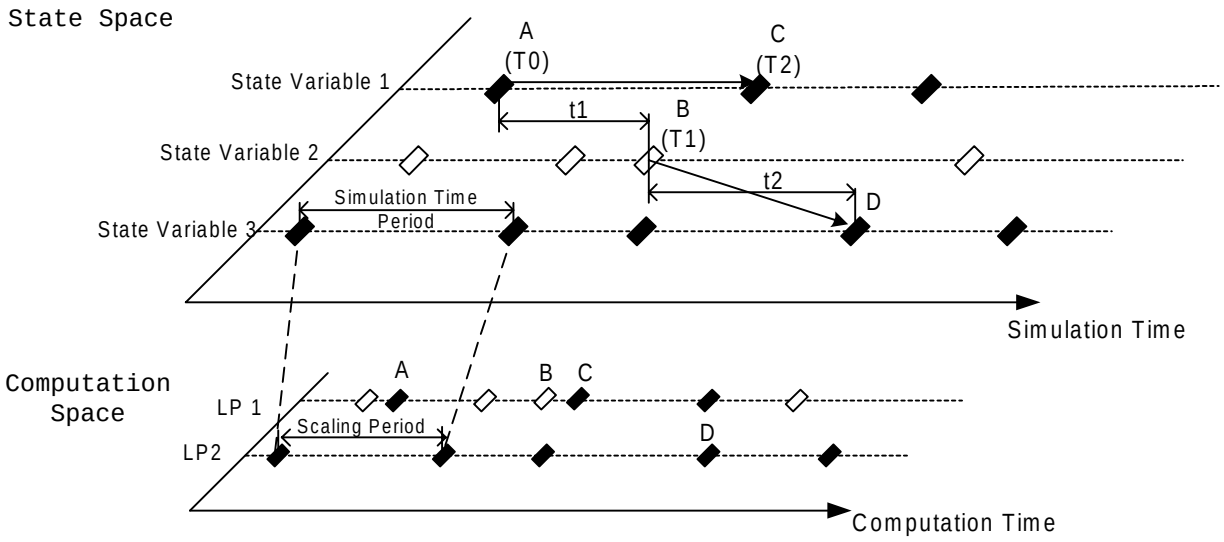


Figure 4.8 Scaling from the state space to the computation space

The second case, where state variables handled by different LPs, can be justified in the same way. Also shown in Figure 4.8, state variable 1 is handled by LP1, and state variable 3 is handled by LP2. Suppose state variable 1 generates an event A at simulation time T0 on LP1, and state variable 2 generates an event B on LP1 t1 time unit after event A. On LP1, event A will be simulated t1/S before event B. If event B causes state variable 3 generating an event D at t2 time unit after event B, according to the observation, no matter what happens, event D will not be processed by LP2 at computation time (T0 + t1 + t2)/S, which is greater than T0/S (when event A is being processed), and (T0 + t1)/S (when event B is being processed). Therefore, event D can only be processed by LP2 later than event A and event B. The correct sequence of

A→B→D is maintained. Actually, this observation states the fact that through such a time scaling mechanism, the simulation procedure can truly reflect the state changing of the physical system in each detail step.

To benefit from above observation, a mechanism is needed to execute the simulation strictly following the sequence in computation space. Conventional parallel simulation methods, either conservative or optimistic PDES does satisfy this requirement, but at a high cost of synchronization. By introducing the specific architecture used to simulate FMS and directly implementing it at the hardware level, it is possible to reduce the synchronization cost so as to achieve a high speedup.

4.5.2 Realization of Time Scaling Simulation Method

The basic idea of the time scaling simulation method is to use the microprocessor's real digital clock, not a virtual clock variable, as the global time to synchronize the simulation. In other words, the time period of an activity eclipsing in the real world are now represented by the amount of clock cycles last on the microprocessor to represent the activity. This is called time scaling. For example, if 1 clock cycle is used to represent 1 minute in the real world, a cutting process lasting for 15 minutes can be simulated by a 15 clock-cycles computation, equivalent to 3×10^{-7} second if the frequency of the clock is 50 MHz. Based on the virtual clock, the timing mechanism is simplified to keep up with the clicking of a crystal clock. During the simulation, all activities are naturally synchronized by the digital clock. This time scaling simulation method provides one advantage in that no additional synchronization mechanism is needed because all events occur exactly in the sequence they should happen with the assumption that there is no communication delay between microprocessors.

The time scaling simulation method is illustrated in Figure 4.9. During the simulation, the activity of processing part A on workstation 1 is represented by several clock cycles eclipsed on microprocessor 1. If, according to the dispatching rule, the next operation for part A is scheduled on workstation 2, after finishing its processing on part A, microprocessor 1 will send a message to microprocessor 2. For microprocessor 1, this out-going message indicates a part A departure event. For microprocessor 2, it is an in-coming message indicating a part A arrival event. The transportation time needed to move the part from workstation 1 to workstation 2 can be

represented by the clock cycles spent to pass the message as well. If there is a part B being processed on workstation 3 at almost the same time period and its next operation is also arranged on workstation 2, microprocessor 3 will send a message to microprocessor 2, too. The arrival sequence of these two messages about part A and B naturally represents the time order of these two arrival events. If under the FIFO dispatching rule, in this case, part A arrives earlier and begins its processing on workstation 2. Later on, part B also arrives on workstation 2, but it has to wait in the queue because the machine is occupied by part A.

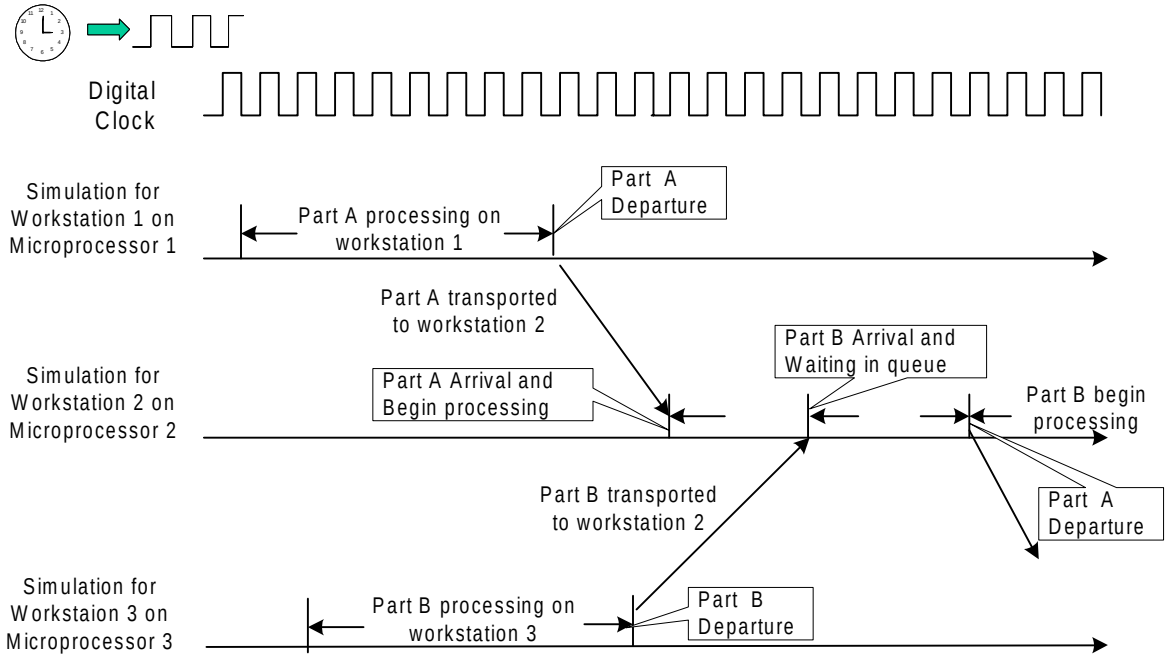


Figure 4.9 Time scaling simulation method

The sequence to process these simulation events is naturally synchronized by the digital clock. Different from any software-based simulation, where the processing routine to handle an event is called after the minimum time is determined among the current event list by the timing routine, the processing routine to handle an event in the hardware-based parallel simulation starts when an event is scheduled to happen. Theoretically, this provides one advantage of the time scaling simulation method that no additional synchronization mechanism is needed, because the sequence of processing events follows exactly the same as that of events occurrence. Moreover,

the well-controlled communication between microprocessors ensures that there are no arbitrary communication delays, nor transient messages among microprocessors. As a result, once a message is received, all other messages received later on must have a stamped time larger than current message, no matter which microprocessor sends those messages. This is true for any event. Therefore, any event can be handled immediately by the microprocessor once received.

Another advantage of the time scaling simulation method is that it is deadlock free. Deadlock free means that the simulation method will not cause deadlock under any circumstances. It is a common recognition that there are four basic necessary conditions required for a possible deadlock. They are exclusive sharing, holding and request, no preemptive control, and circular waiting. If any one of these four conditions can be avoided, deadlocks will not occur. Deadlock is usually a big issue for PDES, especially for conservative PDES, because huge amounts of waiting processes exist, and there is a very good chance that those waiting may form a circular waiting and cause a deadlock. Some mechanisms either to avoid deadlock (by sending NULL messages) or detecting and recovering from a deadlock (by an engagement tree and broadcasting minimum time to all descendent leaves in the tree), must be added to the simulation processes. Therefore, a significant amount of computing power will be wasted in handling the deadlock issue using conservative PDES. In the time scaling simulation method, because the processes to handle the events are started by the event itself, no waiting is needed. Once an event happens, the time scaling simulation method ensure that there are no other events can happen earlier than this event within the whole system, so it can be processed immediately and does not need to wait for other events. The deadlock free feature simplifies the control of simulation execution, which makes the time scaling simulation method more suitable to be implemented at the board level.

Until now, all aspects related to the hardware-based parallel simulation, including the architecture constructing, modeling elements, model executing, and results collecting have been discussed. The next question is how well can the hardware-based parallel simulation perform? To answer this question, some speed analysis of the hardware-based parallel simulation is desirable.

4.5.3 Speed Analysis

The major factors that can affect the speed of the proposed time scaling simulation method are the magnitude of the scaling factor, the frequency of the digital clock, the computing difficulty in representing control logic, and the simulation time period. As far as the frequency of the digital clock is concerned, the larger the frequency that the microprocessor can use, which means the faster the microprocessor can run, and in turn, the faster the simulation can run. For the simulation time period, if a longer simulation time period is studied, the computational time will increase proportionally because more events have to be handled. While the effects of above two factors are relatively easy to determine, it is relatively difficult to find out how time scale and computing difficulty may affect the overall speedup.

To determine the time scale, details about the computational process used to represent an activity must be studied first. Generally speaking, the computational process to handle one simulation event can be divided into four parts: the code to generating random numbers (if needed), the code to implementing control logic, the code to recording simulation data, and maybe a no-operation portion (see Figure 4.10). The no-operation portion is the idle time used to compensate the computing period to the scale of a long interval time period. Thus, the scaling factor can be calculated as the number of the real time needed to finish the activity divided by the total clock cycles needed to complete all computations in each processing routine. It means that the scaling factor is determined by two factors, the level of detail to be simulated, and the execution speed for each processing. If more details need to be simulated, smaller real time intervals must be used. As a result, the scaling factor will be smaller, which means a longer time to finish a simulation run. For the second factor, if a routine can process an event faster, then the scaling factor can be larger, which means less time to complete a simulation run. How to choose this scaling factor may have a big impact on the final speedup of this method. In summary, the bigger the scale factor can be used, the faster the simulation execution can go.

The computing difficulty coefficient is actually an adjustment of the time scale according to how difficult it is to implement the control logic. The consideration here is that while other portions of the computation are relatively constant, the code needed to realize control logic in simulation can vary much, depending on how complex the dispatching rules can be. Obviously, the more complex the dispatching rule is, the more code is needed, and the longer the simulation

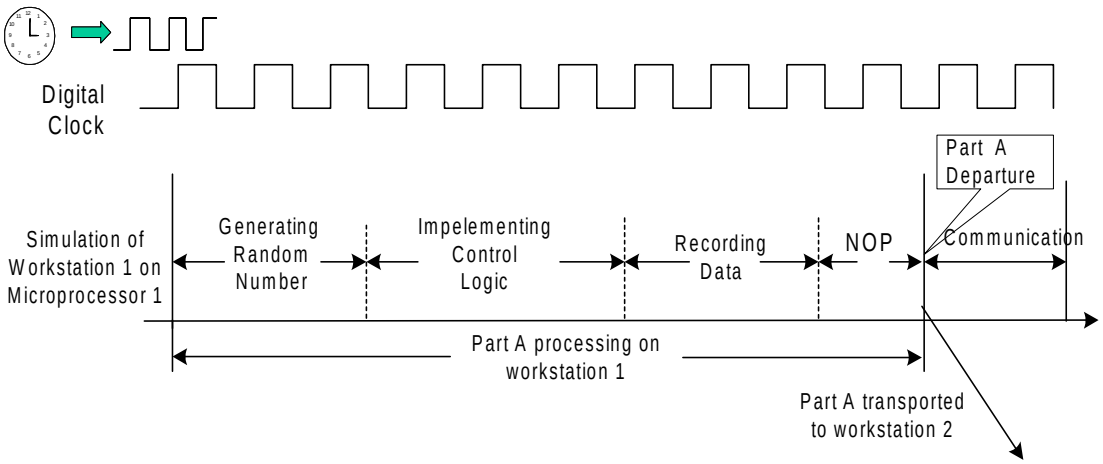


Figure 4.10 Typical computing process used to represent activities in hardware-based parallel simulation

has to run. Unfortunately, it is very difficult to realize complex dispatching rules merely by some hardware circuits. But by using microprocessors as the main computing power, the hardware-based parallel simulation can implement some complex logic in the way of micro-programs at the cost of losing more speedup. In general, the more difficult it is to implement the logic control, the slower the simulation execution can go.

Based on above analysis, the execution time of the hardware-based parallel simulation Ts can be estimated by following formula:

$$Ts = N \times \frac{m \times t}{s \times f} \tag{4.1}$$

where, Ts: computing time needed to execute the simulation

N: replications of simulation

m: computing difficult coefficient

s: time scaling factor

t: simulation time period

f: frequency of the digital clock

Based on this formula, the execution time can be quickly estimated before a hardware-based parallel simulation actually starts. To demonstrate it, simulation of the laboratory FMS described in previous section is used as an example. The time scale factor can be obtained by choosing the minimum time interval of all activities needed to be considered, that is 0.1 minute to measure the width and height of raw materials. If an estimation of 100 clock cycles is needed to finish necessary computation, then the time scaling factor s is equal to 6/100. The computing difficulty coefficient m is estimated about 2 to implement some simple dispatching rules, which means additional 100 clock cycles are needed to implement these dispatching rules. The simulation time period is the same as the software-based simulation (say 256 hours, t = 256*3600 seconds), and the simulation runs for 10 replications (n = 10). Suppose the board uses a 50 MHz clock ($f = 50*10^6$). Based on this information, the estimated execution time Ts = $10*(2*256*3600))/(6/100)/(50*10^6)$, around 6.14 seconds, which is 10.7 times faster than the 72 seconds from the software-based simulation. But this is just a rough estimation about the speed of the hardware-based parallel simulation. To find out the real speed of the hardware-based parallel simulation, more experiments have been designed and carried out.

4.5.4 Modifications on Basic Hardware-based Parallel Simulation Method

Unfortunately, previously proposed time scaling simulation method is only theoretically true for an idea "pure" hardware-based simulator with mass parallel computing power. For the multi-microprocessor architecture based simulator built in this research, because the computing time to realize any activities or logic on a microprocessor needs a sizable amount of clock cycles, the simulator can only provide a pseudo-parallel computing power. Hence, several adjustments to the basic time scaling simulation method must be made in order to implement the hardware-based parallel simulation on a prototype simulator with multi-microprocessor architecture.

The first adjustment is made to the timing mechanism. In the basic time scaling simulation method, the digital clock determines all the time periods. If the program needs to know a time period for an activity, it must read from the digital clock. This requires a very high resolution of the digital clock, leading to a lower scaling factor and lower overall simulation speed. However, for the purpose of timing, exact time points are not important if the sequence of

the events can be determined by other ways. Based on such an understanding, instead of reading the time points from the digital clock, the routines will record the exact time points of every event, and allow the digital clock to perform functions only for synchronization and timing. For example, suppose through the random number generation, the next cutting process for part A is generated to last for 5.235 minutes. Using the basic time scaling simulation method, this 5.235 minutes will be transferred to 52 cycles, if the time scale is 0.1 minute/cycle. After the digital clock clicks 52 cycles, the program will read a 52-cycle period, and instead of 5.235 minutes, 5.2 is actually used in the simulation for statistical data calculation. Because the number 5.235 must be generated first by the random number generation and is known to the simulation program, it can be stored in the memory at that time. After the digital clock clicks 52 cycles, instead of reading the time from the clock, the program can read the time from memory as 5.235 minutes, and uses this exact number for the statistical calculation. Using the accurate number instead of reading from the digital clock helps to maintain the right sequence in generating events and the right final condition in terminating the simulation. Because of the round effect, two events with close time stamps are triggered to happen at the same time. In the central logic, by comparing their accurate time, the program is able to follow the right sequence to handle one event, and then the other. As termination is considered, if using rounded cycles to decide when to end the simulation, because of the accumulation of the round error, it is possible that more events can be generated during a given simulation time period. But by adding the accurate number to an internal time variable, the simulation can be correctly terminated at the end of the given time period, without generating more events. In such a sense, the digital clock is only responsible to synchronize different microprocessors, but not functions as the simulation clock. This adjustment really means to combine the time scaling method with event driven for hardware-based parallel simulation. That is, on each microprocessor there is a discrete event driven simulation engine to advance the simulation locally, and overall simulation advancement among microprocessors is controlled by a global digital clock. This global clock drives the event timers so as to synchronize the simulation clock among different microprocessors.

Such hardware-based parallel simulation with the help of timers is described below. For each type of event, there is not only a logic routine to handle it, but also a hardware timer to trigger when the routine should start. For example, for routine handling parts arrival, there is an arrival timer. This timer is set by the arrival routine to a certain number of cycles that stand for

the interval time between part arrivals. The number is calculated by the time scaling method. After this number is set, the timer will be driven by a global digital clock. When the timer counts down to zero, which means a part arrival occurs, the timer will generate an interrupt signal to the microprocessor. This interrupt signal is translated by a central logic routine to start corresponding arrival routine. The arrival routine will handle current part arrival, either put it into queue or directly pass it to a departure routine. Then the arrival routine will set the next part arrival time spot to the timer, and wait for the timer to trigger next interrupt. This circular process continues with the advancement of the simulation. For each type of event, such as departure and machine broken, there are corresponding timers which are responsible to trigger their routines to start at correct simulation time spots. Such a timer associated hardware-based timing mechanism can be shown in Figure 4.11.

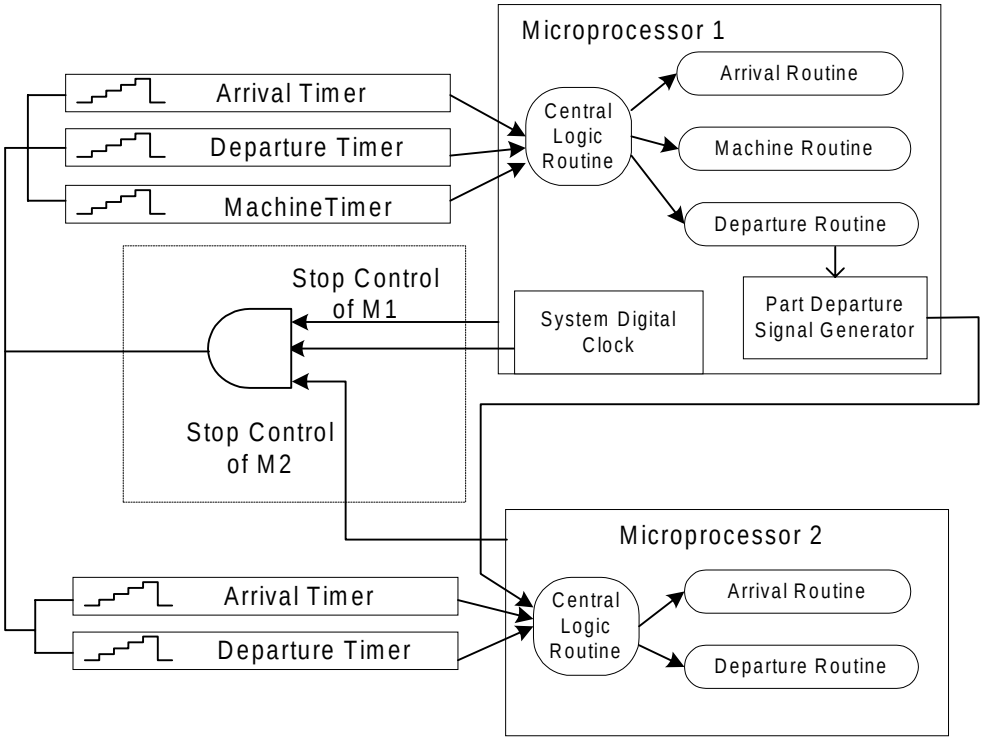


Figure 4.11 Timer associated hardware-based timing mechanism

The second adjustment to the basic time scaling simulation method is adding a stop mechanism. In the basic time scaling simulation method, the time scale is determined by the least time interval between two events, which ensures that any necessary computing can be completed within those cycles, which are calculated from dividing the time interval by the time scale. If the values of all the time intervals are almost identical or relatively close to each other, the digital cycles allowed to process the corresponding events are also close, so that not many NOP operations need to be added into the computing. But in most cases, the time intervals between events may vary in a large range. If the computing for events with the least time interval must be finished within certain cycles, then a lot of NOP operations must be added to the computing process for events with much larger time intervals. As a result, a lot of computing time is idle for the purpose of synchronization. One possible solution is to use some certain time intervals P greater than the least time interval to decide the time scale. Of course, for the least time intervals or any time intervals smaller than this predefined number P, the digital cycles are not long enough for the processing routine to finish its computing. Under such a situation, the simulation process must stop temporarily to allow the computing to finish, and then resume the simulation exactly from that time spot. To stop the simulation really means no new events can be generated. In a software parallel computing environment, it is very hard to realize such a mechanism because the communication is controlled not by the simulation processes but by some other communication control processes. Suppose one microprocessor wants to stop the simulation, it must broadcast a privileged "STOP" message to all other microprocessors. This message may reach different microprocessors at different moment because of the arbitrary delay in the communication network. As the result, there is no guarantee that the overall simulation can be stopped with no more new events generated in the whole system. Most likely, each microprocessor will stop at different time spots, causing errors when the simulation resumes. Fortunately, it is not difficult to implement a stop mechanism on the simulator. Because the processing of an event is triggered only by the signal from the system digital clock, if the digital clock stops, the whole simulation will stop at that time spot. The idea is to give each microprocessor the ability to control the digital clock. Whenever necessary, any microprocessor can stop this digital clock so as to stop the whole simulation.

In implementation, instead of connecting the system digital clock directly to all timers, the clock first inputs into an AND gate with the "STOP" control port on all microprocessors,

then the output of this AND gate is used to drive all the event timers, also shown in Figure 4.11 as well. Under normal conditions, all microprocessors will set their STOP control output to '1', the output of the AND gate will be a clock wave. If any one of the microprocessor wants to stop the simulation, it only needs to write '0' to its STOP control port, and the output of the AND gate will always be '0', which is not a clock wave. As a result, all timers will stop, and no new processing routines will be triggered. In this way, the whole simulation can be viewed as just stopping at its current time spot. To resume the simulation, the microprocessor only needs to write a '1' to its STOP control port, then the simulation will continue from where it stopped. Such a stop mechanism can be used to solve the problem when the interval period of the timer is not long enough to let a routine finish its computation during that period.

4.6 Difference Between Hardware-based Parallel Simulation and PDES

Obviously, Parallel Discrete Event Simulation (PDES) can also be programmed on the proposed simulator. Before describing how to implement PDES on the simulator, a brief introduction about PDES is introduced first. The basic idea of PDES is to decompose a simulation model into submodels and distribute the simulation computation on multiple different logic processes (LP). These logic processes can be executed on different processors simultaneously. Processes communicate with each other by means of message passing. Different from conventional sequential discrete-event simulation, which utilizes a global clock variable to advance the simulation, there are no shared variables, nor central control among processes in PDES. The physical system being modeled is viewed as being composed of a number of interactive physical processes (PP). To model the physical system, each PP is represented by a logical process (LP). The interactions between PPs are represented by message passing between LPs. All actual interactive activities between PPs at a certain time are simulated by time-stamped event messages passing between their corresponding LPs. The time attached to the message stands for occurrence time of that event in the physical system. Each LP has its own local simulation clock associated with it. This local clock variable denotes how far the sequential portion of the simulation on this LP has been processed. LPs may advance the simulation to different points in time. By comparing the stamped time to its local clock, an LP is able to check the order of in-coming events from other LPs. But a violation of the causality principle might

occur, that is, an in-coming event with a stamped time earlier than the current local time, then some correction mechanism must be employed in order to maintain the right sequence of events to be simulated and generate the correct simulation results.

By ignoring special hardware features, the proposed simulator provides a suitable hardware platform for PDES. Through the space parallel decomposition method, a PDES simulation model for a flexible manufacturing system can be divided into several submodels, in which each submodel simulates portions of the FMS, such as a machining center, a load/unload station, or a material handling unit. From the viewpoint of simulation, each submodel contains a disjoint subset of the model state variables. To process the simulation, each submodel is mapped to a logic process, which is responsible for computing the values of the state variables for the corresponding submodel over the simulation time period. To run the PDES on the simulator, such LPs can be computed on one micro emulator, or several LPs on one micro emulator, determined by the granularity of the simulation model. For the PDES method, either the conservative approach or the optimistic parallel simulation approach can be implemented on the simulator. The conservative approach strictly avoids causality errors at any time point during the simulation by implementing a safe time mechanism. The safe time is used to ensure that only events before the safe time point can be processed by its LP. Although it is relatively easy to maintain the causality constraint in this way, the efficiency of parallel simulation is impaired because most time processors will stay idle waiting for the slowest computation for the least time event to be finished first. A detailed conservative PDES experiment running on the prototype simulator will be introduced later in the programming section.

An advantage of running PDES on the simulator is that it does not have problems related to the unreliable and arbitrary communication delay. In common asynchronous parallel computing environment, instead of being controlled by the simulation process itself, message passing is really controlled by some independent communication processes. Once a simulation process issues a message-sending request to the communication process, it loses the control to the message. Messages might arrive their destinations arbitrarily, or there are transient messages, which have been sent but have not yet been received by the destination processor. Special efforts, such as waiting barriers, have to be added to deal with these problems. This means more waiting and more chance to have deadlocks. However, on the proposed simulator, integrating all microprocessors on one board provides a tightly coupled and well-controlled communication

links between these microprocessors. It can be assured that the sequence of arriving messages are following the same order of sending, which is very important to the PDES because the sequence of messages actually implies the order of simulation time. Moreover, because the communication is controlled by the simulation processes itself, some mechanisms can be adopted to ensure that there are no transient messages in the communication network. These advantages can eliminate a huge overhead to handle communication problems for parallel discrete event simulation mechanisms. For PDES running on the simulator, synchronization among micro emulators to preserve the causality constraint while exploring the internal parallelism as much as possible is still a difficult task to undertake, mainly because each LP has its own local clock and those clocks may advance to different time points. Another problem is to avoid the deadlock that can be caused by the PDES method itself because there exist many types of waiting during the simulation.

A summary of previous discussions shows that there are two simulation methods that can be implemented on the proposed simulator. The first one is the conservative PDES running on proposed simulator. The second one is the modified hardware-based parallel simulation on this multi-microprocessor based simulator, which is also the focus of this research. These simulation methodologies are different from general software-based sequential simulation method and the ever-growing conventional PDES.

Conventional PDES usually runs on a general parallel computing platform, either a distributed computer network or an advanced parallel computer. Although conventional PDES has the flexibility to build simulation models for various kinds of physical systems, it suffers from the uncontrolled communication network, which results in a big waste of computing power on handling the synchronization and deadlock problems. There are some essential differences between the hardware-based parallel simulation method and the conventional PDES. These differences involve the method used to build the simulation model, the representation of interactions between microprocessors in the simulation run, and the timing mechanism.

First of all, the method used to build the simulation model is different. In PDES, a simulation model is developed by using software components to represent the machinery. In the proposed simulator, the simulation model is determined by the architecture of the simulator

itself. By carefully placing the micro emulator modules in exactly the same pattern as the physical layout of the real FMS, the simulator itself constitutes the simulation model.

For the representation of interactions between microprocessors, PDES relies on a message-passing mechanism, all the activities, such as part arrivals and departures to other machining centers are merely represented by receiving messages and sending messages. In the hardware-based parallel simulation, besides messages which carry the detail information about a part, the activity of a part arrival or departure is really triggered by an outside signal. It is this signal that represents the part arrival or departure, and starts the corresponding simulation routines. It is the digital clock that controls the generation of these digital signals, not the simulation routines. In such a sense, parts are being represented by digital signals, not by messages.

For the timing mechanism, in PDES, timing means to use some time variables (either global or local) and advance these time variables by a timing routine. The problem is that until the LPs can ensure that it is "safe", which means no more messages received in the future will have a stamped time less than current local time, it cannot advance its local time and has to wait for other microprocessors telling it what the "safe" time is. Such a method wastes a considerable amount of the computing time on waiting. In an even worse scenario, it can easily cause deadlock and cannot finish the simulation at all. In the hardware-based parallel simulation, timing is achieved by using the digital clock on the board. It is his digital clock that triggers the start of processing routines to handle different events. The function of this digital clock is very similar to a global time variable used in PDES. All microprocessors synchronize their local time to this digital clock. If not triggered, the local time will remain its previous value. During the simulation, local times can be advanced to different time spots, and they will synchronize to each other whenever there is an interaction between any two microprocessors. By using such a digital clock independent from the simulation processing, it can ensure that no new event can be generated with a smaller time stamp within the whole system after an event is already generated.

Differences between PDES and hardware-based parallel simulation are summarized in Table 4.2. Generally speaking, the hardware-based parallel simulation is fast but system specific, while PDES is flexible for modeling different systems but relatively slow.

Table 4.2 Differences between PDES and hardware-based parallel simulation

	PDES	**Hardware-based parallel simulation**
Modeling	Software components	Hardware components
Entity Representation	Virtual variables and messages (Parts, events)	Real digital signals and messages (Parts, events)
Logic Control and Timing	Software Program	Micro-program plus on-board digital clock, or electronic logic circuit (for customized simulation chip)
Communication	Loose and arbitrary	Tight-coupled and well-controlled
Advantages	Graphic interface, powerful analysis tools, flexible	Possible speedup, fast to realize true on-line real-time simulation
Disadvantages	Slow, complex modeling	Less flexible in reconfiguration

CHAPTER 5

IMPLEMENTATION EFFORTS

Implementation efforts have focused on building a multi-microprocessor prototype simulator using DSP technology. The prototype simulator employed a combination of shared memory and parallel I/O ports architecture. Because such architecture lacks the massive parallel computing capability for implementing the idea time scaling and driven method, the modified hardware-based parallel simulation is programmed instead on such a prototype simulator. The main difference between this modified method and the idea method is that on each microprocessor a local discrete event driven engine is used to simulate all activities, instead of using pure timing cycles. The synchronization among microprocessors still follows the mechanism using a global digital clock. In this chapter, the development of a prototype simulator and program for conservative parallel simulation and hardware-based parallel simulation are described in depth. Other implementation issues, such as the development cycle, cost, and flexibility of the simulator are also briefly discussed.

5.1 Building a Prototype Simulator

Based on the hardware-based parallel simulation methodology, one prototype simulator has been developed. The prototype simulator was specifically designed to simulate the laboratory FMS described in Section 3.1, by realizing an architecture combining the shared memory based architecture and the I/O port based architecture. Figure 5.1 shows the framework for such architecture and Figure 5.2 depicts a layout of the developed prototype simulator. Overall, the prototype simulator used 4 Texas Instrument DSKs (DSP Starter Kit), connected to each other through dual-port SRAMs and serial communication ports. The dual-port SRAMs serve as shared memory between two DSKs. In addition, there is a direct link between two DSPs, representing a part routing path between workstations in simulation. Such a combination provides an applicable solution while maintaining the basic capability to simulate the laboratory FMS. As the board was developed under the one-to-one mapping, each workstation in the

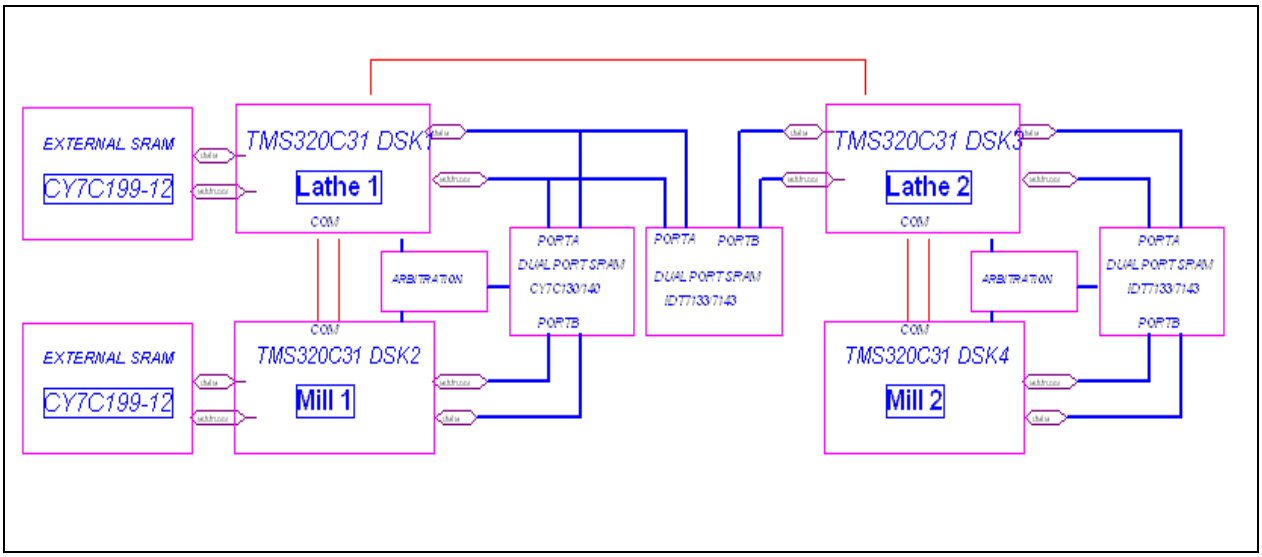


Figure 5.1 Frame of the prototype simulator

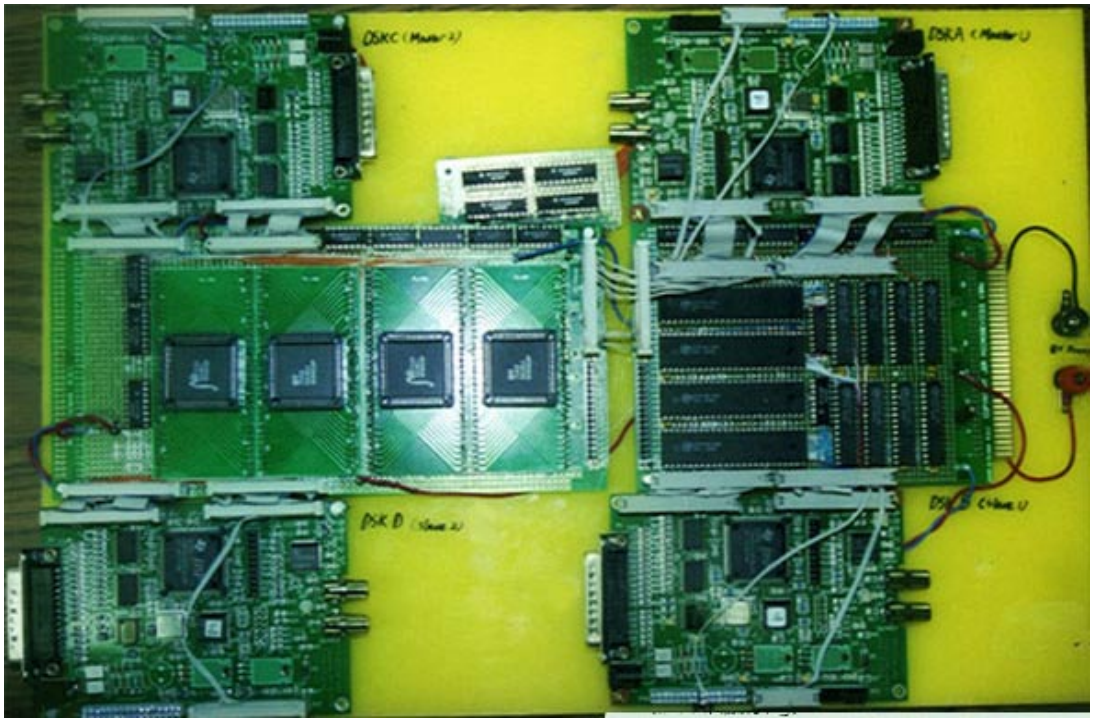


Figure 5.2 The 4-DSP prototype simulator

laboratory FMS is mapped to one DSK on the board. DSK 1 is responsible for mimicking Lathe 1, DSK 2 is responsible for mimicking Mill 1, and so on. As shown in the framework, the layout of the board is similar to the physical layout of major components in an FMS.

A Texas Instrument TMS320C31 DSK board is used as individual micro emulator on the simulator. The DSK board is chosen because the cost for each board is low (about $99 for one), and it leaves a large space to modify the board according to our own design. On each DSK board there is a TMS320C31 DSP chip, running at 50M Hz. The most important features of TMS320C31 are listed as follows:

- High performance 32-bit floating-point Digital Signal Processor TMS320C31-50, 40-ns instruction cycle time, 275 MOPS, 50 MFLOPS, 25 MIPS
- Harvard-architecture, with two identical external data and address buses. Bus lock/unlock signals are available to support shared memory systems.
- Two 1K × 32-Bit single-cycle dual-access on-chip RAM blocks
- On-chip memory-mapped peripherals including, one serial port, two 32-bit timers, one-channel Direct Memory Access (DMA) coprocessor for concurrent I/O and CPU operation
- Two address generators with eight auxiliary registers and two Auxiliary Register Arithmetic Units (ARAUs)
- Parallel Arithmetic/Logic Unit (ALU) and multiplier execution in a single cycle
- Block-repeat capability
- Zero-overhead loops with single-cycle branches
- Conditional calls and returns
- Interlocked instructions for multiprocessing support
- Bus-control registers configure strobe-control wait-state generation

More details about TMS320C31 and the DSK board can be found in its technical reference [101]. Basically, TMS320C31 is within the same product series as TMS320C40, servicing as a low-end DSP in the series. Compared to TMS320C40, TMS320C31 has the same computing power (both at 25 MIPS) but less communication ports and DMA channels.

Some peripheral electronic circuits, such as those for the stop mechanism, have also been developed on the board. Appendix B documents all schematic diagrams for the prototype simulator. In the design, 4 TMS320C31 DSKs were used. Cypress' CY7C130/140 (1K word) and IDT's IDT7133/7143 (2K word) were the dual-port SRAMs used as the shared memory on the board. The read/write cycles time for both CY7C130/140 and IDT7133/7143 are 35ns. Because this access time is very close to read/write cycle time (40 ns), to ensure correct read/write operations to the shared memory, a one-cycle wait is automatically added to all read/write operations when addressing the shared memory.

To avoid collision on same address location, the dual-port SRAMs themselves provide on-chip arbitration logic to control access to same locations. There is no problem when two DSPs read-read or read-write simultaneously to the same memory location, but write at the same time to the same location can cause collision problem. When two DSPs try to write to the same memory location at the same time, the on-chip arbitration control will decide which port of the dual-port SRAMs can gain the access to the location, and which port will lose the competition. A busy signal will be automatically generated by the arbitration control to the losing port. This busy signal is linked to the $\overline{HOLD}$ pin of TMS320C31, which can prevent the DSP from finishing its current read/write operation until the busy signal is released by the other DSP. Besides, a hardware mailbox on the chip can generate interrupts to inform other microprocessors that the message is ready.

As proposed, the shared memory is used only to exchange information about parts or events between DSK boards, such as the types of part, part arrival time, etc. Different from the general PDES, a part arrival or departure is represented by a digital signal sending by DSPs, rather than those messages themselves carrying only information about the parts. The signal sending is initiated by using the Timers on the microprocessor, controlled by a global digital clock. The global digital clock is the system clock on one DSP. Besides the shared-memory, each DSP has 2K RAM on chip as local memory. In addition, 32K word external RAM is added to DSPs in case the 2K RAM is not big enough for statistical data storage. If disregarding those special circuits to facilitate the simulation, the board can also provide a suitable parallel computing platform for conventional parallel or distributed simulation.

Using the hardware simulator alone cannot finish the simulation task. To simulate the FMS, suitable simulation mechanism, which can take advantage of those features of specifically designed hardware, has to be developed. Based on such a mechanism, assembly programs have to be programmed and executed on the prototype simulator to eventually achieve the rapid simulation for FMS.

5.2 Programming for the Simulator

Most subroutines programmed for the simulator are very similar to general parallel simulation methods. These subroutines deal with the main flow control, arrival events, departure events, random number generation, and statistical reporting. They can be easily learned from some popular simulation books, such as Fujimotto's *Parallel and Distributed Simulation System: Algorithm and Applications* [37]. Only the data structure and critical subroutines for timing, routing and robot control are introduced here. They basically distinguish this hardware-based parallel simulation from other simulation methods. Details about other subroutines are presented in Appendix C.

5.2.1 Data Structure and Variables

The data structure and variables can be divided into four groups: global control variables, structural data, statistical variables, and shared variables. Figure 5.3 gives a complete view of the data structure used in this work. According to the requirement of different simulation tasks, only a portion of the set may be used in any particular program file.

1. Local Control Variables

Local control variables are mainly used to control local simulation procedure. These variables include:

Time: used to keep track of the local simulation clock. It is determined by choosing the minimum time from all events in the event list. It can be used in any subroutines whenever current simulation time is needed. It is also provided by the master microprocessors to slave microprocessors as safe time.

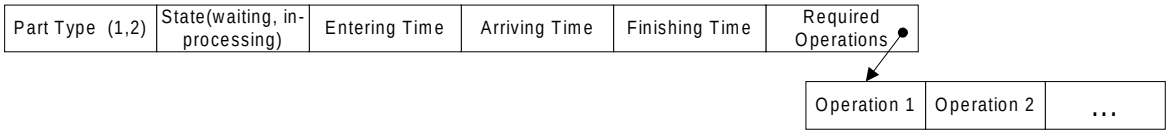
1. Important Local Variables

Time : current simulation clock
LastTime : simulation clock of last event
EndTime : time when simulation ending

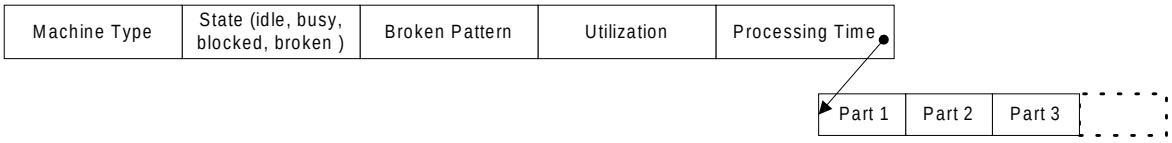
NextEvent:: next type of event to be processed
MeanArrival : mean time between interarrival parts

2. Structural Data

Part, used to organize all in-processing and queued parts

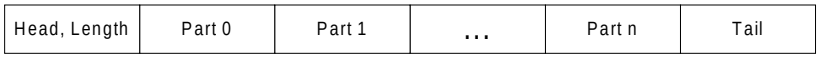


Workstation, used to store the status and statistical records of the workstation

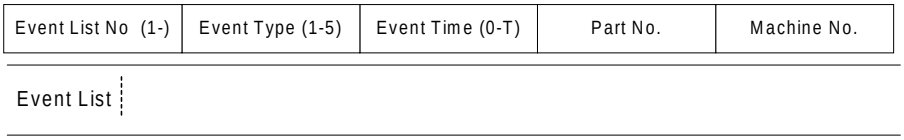


Robot, used to store the status and statistical records of the robot, travel time is calculated based on a position matrix

Queue, for conveyor, input queues and working-in-process queue, basic element is made of part record



Event, used to construct event list (organized as a linked list)



3. Statistical Variables

NumInQ : number of parts in queue
SumPart : total throughput
TotNum : total arrivals

AveInQ : average in queue
AveStay : average stay in system
AveDelay : average delay of parts

4. Shared Variables

flag

PartArrival : signal of part sending
PermitSend : permission to send part
RequestSend : request to send part
MxDown : workstation status on DSP x, x= 1, 2, 3, 4
MxQFull : WIP queue of workstation x full, x=1, 2, 3, 4

Message

MXSafeTime : safe time on DSP X, X=1,2, 3, 4
M1_M2Event : message on routed event
M1_M2Part : message on routed part

Figure 5.3 Complete set of data structure

LastTime: used to record the last time spot before current event. It is mainly used for statistical computations.

EndTime: used to define the length of the simulation period.

NextEvent: indicating what is the next event to be processed.

MeanArrival: mean time between arrivals of different part types. It determines the frequency of parts input to the FMS.

2. Basic Structural Data

Although assembly language is the basic tool to program the simulator, adopting the philosophy of object-oriented design into the programming can help to better organize the data and program flow. Hence, structural data elements are developed to describe the characteristics of parts, workstations, robots, and queues. Most of the simulation parameters can be dynamically changed during the execution according to the data gathered from sensors which continuously monitor the running status of simulated FMS. These structural data elements are:

Part: data structure used to describe the part object in systems, either in processing or waiting at temporary storage places. It is composed of attributes recording information about individual part, such as part type, processing state, operating requirement, entering, starting, and leaving time. It is also the basic storage element for queues.

Workstation: data structure used to describe individual machining center in the FMS. This data structure groups such attributes as equipment type, working state, processing time for different types of parts, broken pattern (Mean Time Between Failure and Mean Time To Repair), and current utilization. The processing time for different types of parts are machine related, that is different machines may have different processing times to perform the same operation for the same type of part. The broken pattern of individual equipment shall be determined by its historical behavior. The utilization attribute can be used to implement dispatching rules regarding the dynamic usage of workstations during simulation.

Robot: data structure used to describe a robot. This data structure provides basic information about the operation of a robot. It collects such attributes of a robot as type, running state, current position, next available time, and utilization. In this research, a robot is simulated as a resource, competing by various operations. Therefore, when one robot is designed to

serve two or more workstations, this data structure must be stored in corresponding shared memory so that all microprocessors can access the data and share the control of this robot.

Queue: data structure used to simulate conveyors and working-in-processing storage, such as an index table. The basic attribute of this structure is the current length of the queue, the limitation of the queue, and followed by the body of the queues, which is composed by a list of part structure.

3. Statistical Variables

Statistical variables temporarily accumulate the value of interested parameters during the execution of simulation. At the end of simulation, report subroutines will generate final simulation results based on the value of these statistical variables. Examples of such variables are the time-weighted summation of parts waiting in queues, staying in the system, or delay in processing. Statistical variables for the final results include the total arrival of parts, throughput of parts, average number in queues, average stay in system, average blocked time, etc.

4. Shared Variables

Shared variables serves as the communication channels among microprocessors. They are unique because both parallel simulation and hardware-based parallel simulation need them to synchronize programs concurrently running on different microprocessors. Thus, these variables must be stored in the shared memory so as to be accessible to both microprocessors. Between any two microprocessors linked by the dual-port SRAM, there should be a set of such shared variables for the purpose of communication. There are two types of shared variables: flags and messages. Flags are mainly used by timing routines to synchronize the execution of programs, while messages carry the information about routed parts and corresponding routing events. Some typical flags and messages used in the programs are listed as follows:

PartArrival: flag indicating that a part routing between two microprocessors happens.

PermitSend: flag indicating that receiving microprocessors finish processing the previous routing event, and allow sending microprocessors to send another part.

RequestSend: flag indicating that sending microprocessors currently have parts to send out, and wait for acknowledgement of receiving microprocessors.

MxDown: flag representing that workstation No. x currently is broken.

MxSafeTime: message of safe time. Local simulation on slave microprocessor x cannot advance beyond this safe time. It is provided by the master microprocessor, and really depends on the possible leading time for different simulation scenarios.

M1_M2Event: message of routing events. It carries information about occurrence time of the event, which part and which workstation is involved in the event.

M1_M2Part: message of routed parts. It carries information about individual routed part, such as type, operations requirement, and entering time of the part.

5.2.2 Timing Subroutines

Timing subroutines mainly distinguish various simulation methods. In this section, timing routines for the conservative parallel simulation and hardware-based parallel simulation are compared. In both methods, timing routes basically perform two functions: to control the execution and to synchronize local simulation clocks among DSPs. The difference is that conservative parallel simulation deploys a safe time mechanism, while hardware-based parallel simulation adopts a hardware timing mechanism. Timing subroutines for a 1-workstation simulation scenario are used as an example to disclose such difference, and Appendix C lists all assembly programs for following discussions.

Figure 5.4 depicts the flow chart for the timing subroutine of the conservative parallel simulation. The conservative parallel simulation is executed on 2 DSPs, DSP 1 and DSP 2. DSP 1 is in charge of simulating arrival events, and DSP 2 is for simulating departure events. Because any departure is caused by an arrival, in such a situation it is easy to point out that the safe time for DSP 2 should be the next arrival time on DSP 1. Any departure events on DSP 2 happening before this safe time can be processed concurrently with DSP 1. But DSP 2 cannot process departure events with a stamped time larger than this safe time because DSP 1 might later send it an event occurring earlier than what DSP 2 has processed. Vice versa, the safe time for DSP 1 is the next departure time on DSP 2. In this way, the safe time mechanism synchronizes local simulation clocks on these 2 DSPs. Moreover, DSP 1 needs to ensure that DSP 2 has correctly read in the information about the last routed part before it can send out a new part.

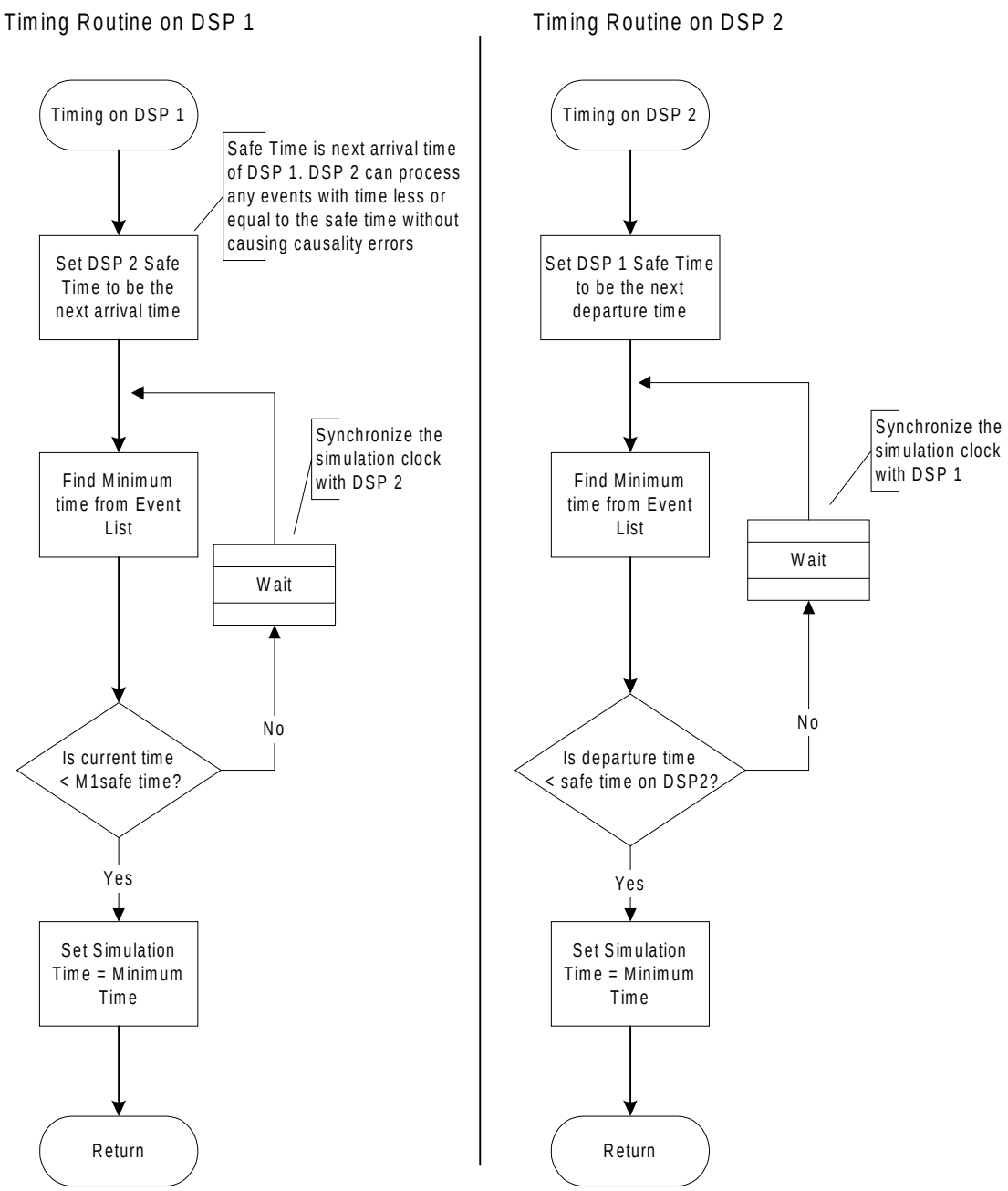


Figure 5.4 Timing subroutines for conservative parallel simulation on DSP 1 and 2

Overall, the timing subroutine for the conservative parallel simulation goes as follows. In the timing subroutine on DSP 1, the program first sends its next arrival time to DSP 2 as the safe time for DSP 2. Then it finds out the minimum time and so the next event to be processed from its local event list, which does not include all departure events on DSP 2. After that, the program will ensure it is safe to process any event by checking whether this minimum time is less than the safe time given by DSP 2. The safe time for DSP 1 should be the next departure time on DSP 2. If it is true, then DSP 1 can process current event in parallel with DSP 2. If not, DSP 1 must wait until DSP 2 processes to a later time spot and gives a larger safe time. The same checking and waiting are performed on DSP 2 as well. Concurrency is achieved when both DSPs can perform their computing simultaneously, but at the cost of wasting time on message-passing and checking for the purpose of synchronization.

Another problem of such a safe time mechanism is that it is difficult to determine the suitable safe time when synchronization with 2 or more DSPs are involved. An even worse problem is that the program flow can quickly grow to a complicated mess with an increasing size of the simulated system and number of DSPs used to do the simulation. Figure 5.5 shows the flow chart for using 4 DSPs to simulate a 4-workstation FMS based on the safe time mechanism. Compared with the 2 DSPs case, this flow chart is much more complex as 1 DSP must synchronize with the other 2 DSPs. Embedded waiting has to be carefully designed because they can easily cause deadlocks by their entangled waiting.

In the timing subroutines of the modified hardware-based parallel simulation, instead of using a safe time to synchronize the simulation on different DSPs, a hardware timer driven by the on-board digital clock is developed to perform the same function. Figure 5.6 shows the timing subroutine on DSP 1 for simulating the same 1-workstation scenario. As stated in section 4.5.4, for each type of event, there is a hardware timer to trigger when an event should start. Therefore, the synchronization is achieved by using a single digital clock for all timers, and the logic timing routine is only responsible for setting up timers. As shown in the flow chart, the program must first check whether the computation for the last event is completed or not. If not, it must halt all timers using the “STOP” mechanism. Timers will be resumed when the event is completed later on. If yes, the program will set up the timer to the value of clock cycles calculated from the time scaling. After initiating this timer, the digital clock will count it down. When the timer goes down to zero, it means that an event occurs at that time point.

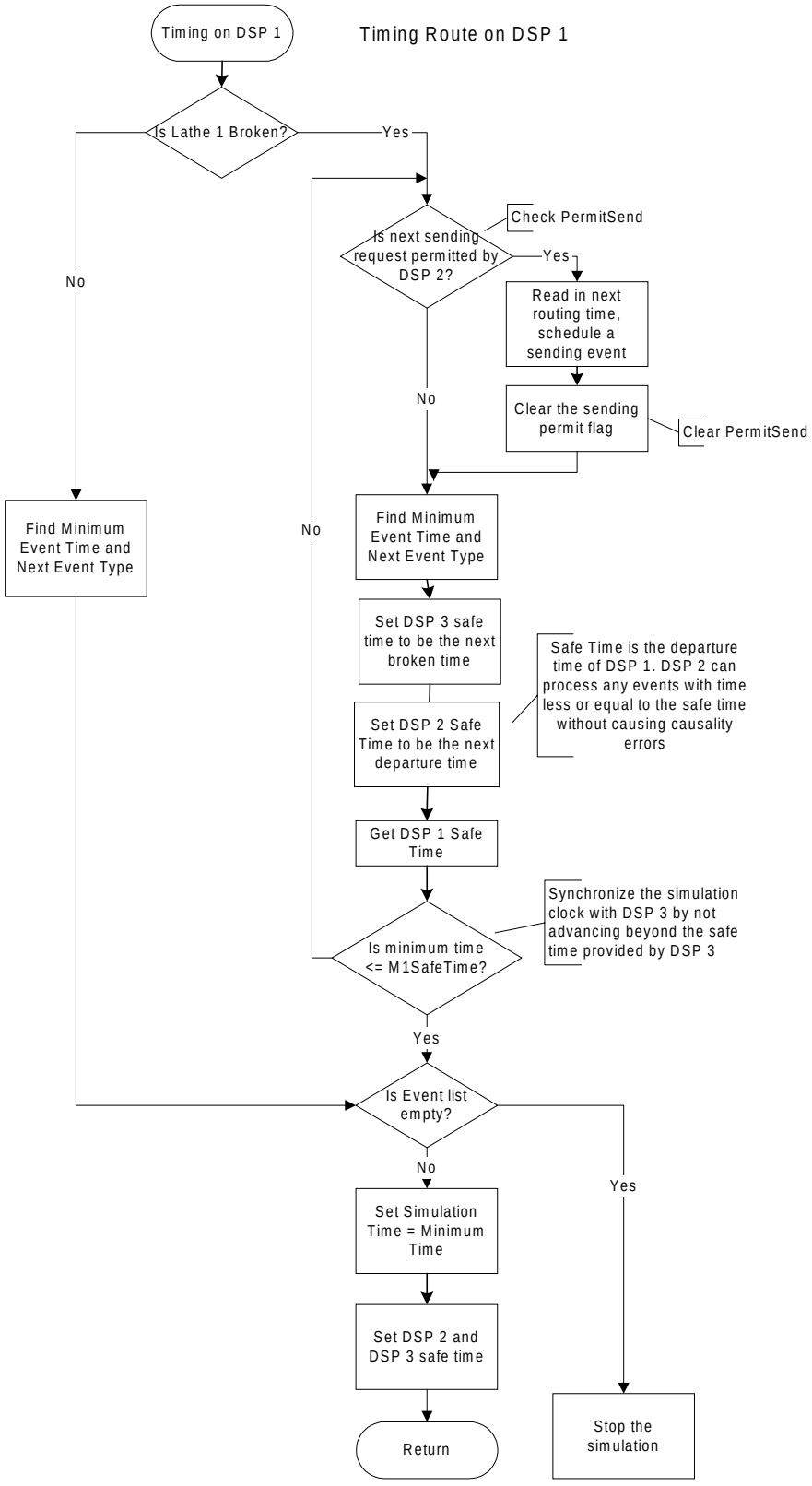


Figure 5.5 Timing subroutine for conservative parallel simulation on DSP 1 (4 DSP board)

The timer then invokes an interrupt service to start processing the event, either as arrival, departure, or machine broken. On DSP 2, the procedure is basically the same. One minor difference is that because DSP 2 is responsible for simulating the workstation, it should continuously inform DSP 1 about the running status of workstation. Compared with the conservative parallel simulation, the modified hardware-based parallel simulation timing routine has a relatively simple logic control so as to save some time in waiting and avoid potential deadlock.

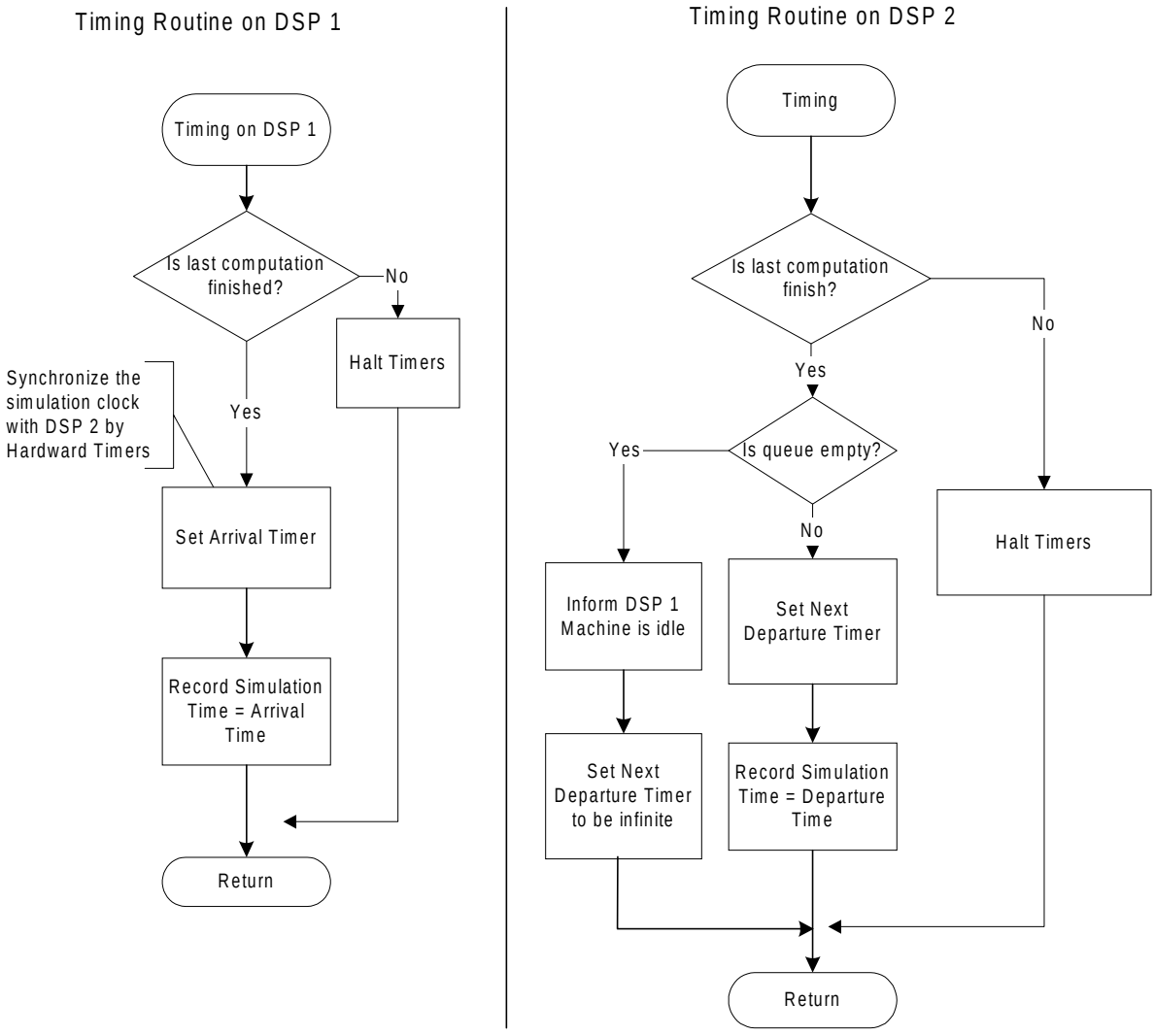


Figure 5.6 Timing subroutines for modified hardware-based parallel simulation

An obvious physical limitation of such a timing mechanism is that the maximum type of events that a program can handle is limited by the number of available timers. Because there are only 2 timers inside those DSPs on the prototype simulator, it is merely enough to simulate a 2-workstation scenario, but not for a 4-workstation case.

5.2.3 Routing Routines

Part routing between DSPs are realized by combining the direct digital link and software sending/receiving subroutines. On the hardware side, there is a direct link from a master DSP to a slave DSP. Whenever there is a part to be routed, a master DSP sends out a "1" pulse to a slave DSP standing for a part arrival. Although in this prototype implementation only one line from master DSP to slave DSP is wired, multiple lines can be wired in a commercial simulator product. On the software side, part routings between DSPs are simulated by sending or receiving subroutines. Figure 5.7 illustrates the logical flow of a sending and a receiving subroutine. Before sending out the digital pulse, the sending subroutine on master DSPs writes all information about this part into the shared memory as a message, including the type, operation requirement, and entering time of this part. On the slave DSPs, receive subroutines are in charge of reading this message passing from the shared memory for the coming part. This receiving procedure is triggered by a digital pulse sending from master DSPs. A flag PartArrival is also associated with this sending-receiving activity. The purpose of this flag is to ensure that the master DSP does not overwrite the message with a new message before the slave DSP can read the message into its buffer.

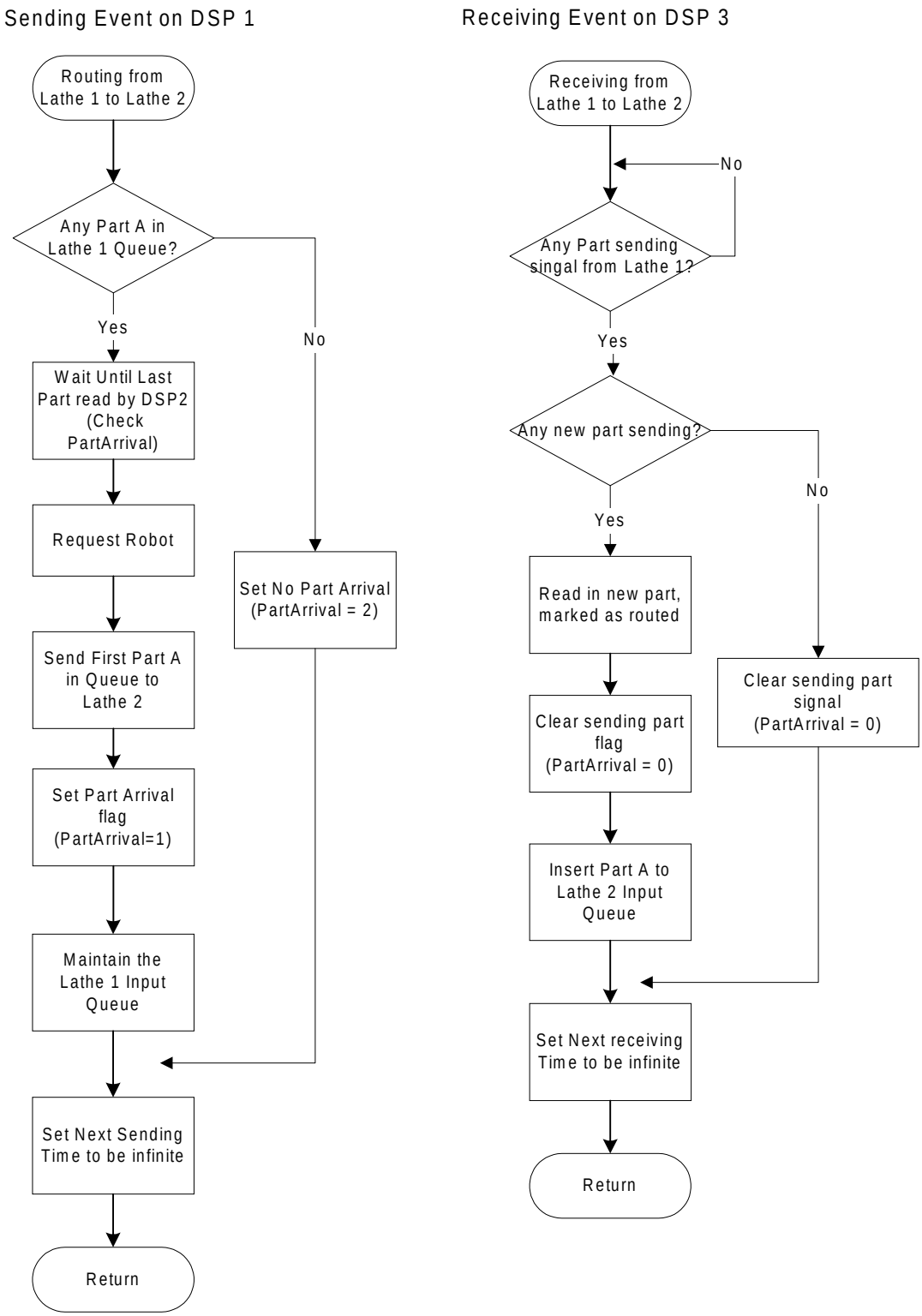


Figure 5.7 Sending/Receiving events for part routing

5.2.4 Robot Control Routines

Robots are simulated as one kind of resource in this work. Such a simplification can help divide the tremendous computation load for simulating a robot among DSPs. If a robot is shared by different workstations, all involved DSPs to simulate these workstations are granted partial control for this robot. For example, if one robot is allocated to serve two workstations in FMS, then a robot control routine shall run on each of two DSPs dedicated to simulate these two machining centers. Figure 5.8 depicts the flow chart for such a control routine. When a workstation needs either loading, unloading, or transportation service, it first requests a robot by calling the robot control subroutine. If the robot is currently free, traveling time is returned.

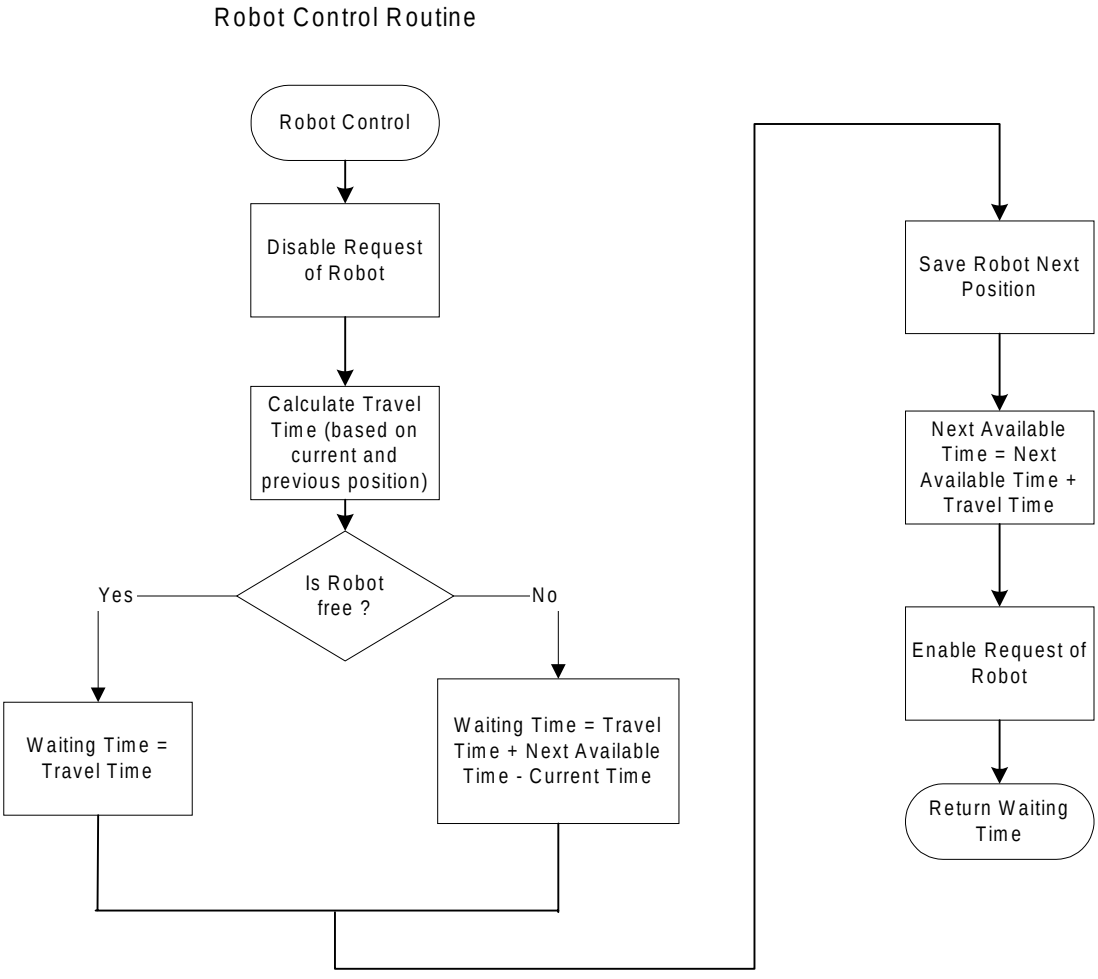


Figure 5.8 Robot control routine

If not, traveling time plus the waiting time is returned. Traveling time is obtained from a distance matrix, which stores all traveling times between any two places within the FMS. Waiting time is equal to the next available time of a robot minus current time. Next available time is when the robot should be free for the next request. All parameters to control a robot are stored in the shared memory so that both DSPs can access and control this robot.

5.3 Other Related Issues

Besides technical issues, there are some concerns about other related issues when the hardware-based parallel simulation mechanism is really applied to industrial applications. These issues discuss development cycle, cost, and updating in real industrial applications. Although each simulator is customized in design to a specific FMS, the general development procedures are very similar for different simulators.

A development procedure usually follows these three phases: design, produce, and install. Because the layout of the simulator must mimic that of the simulated FMS, structure design of the simulator must wait until completion of final design for the FMS. For example, parameters, which determine the basic structure of the FMS, such as how many machining centers are used in the FMS, how AGV path network links these workstations, how the temporary storage (conveyor or buffers) is located within the FMS, etc., have to be determined before designing the simulator as they determine the structure of the simulator. Total workload of design task is largely affected by system size and operating difficulty of the FMS. As far as system size is concerned, it is much easier to design a simulator for a 4-workstation FMS cell than for a 10-machining center FMS. Operating difficulty is mainly affected by how many unique features are used in the FMS. After design, the actual manufacture and installation of the simulator can be carried out concurrently with the construction process of the real FMS. Through this research, a rough estimation can be made for the total workload needed to implement such method. The design for the prototype simulator took nearly 6 months. Then about 4 months were spent on wiring, soldering, and testing the board. After that another 4 months were spent for programming and debugging on the simulator. Overall, a total of 1.17 man-year was spent on finally realizing this simple simulator. This number is only a rough estimation of the research efforts for such

methodology. As for a large production, a much shorter developing time can be achieved because of the learning curve effect and more off-the-shelf modules are developed.

The material costs for such simulators are relatively low. Using this prototype simulator as an example, the cost for 4 TMS320C31 DSK boards is $396 ($99 each), the cost for 10 dual-port SRAMs is about $200 ($20 each), and other comprehensive costs for board including other microchips and wires is no more than $100. The total material cost for this board is no more than $700. Compared to an expensive general purpose parallel computer, such a hardware-based simulator provides a cost-effective solution to provide a parallel computing platform on shop floors. Additional costs are associated with all developing phases, such as design, product, installation and maintenance phases. Because the workload for these phases varies in a large range, this portion of cost also changes a lot with respect to the workload.

Another big concern on such a simulator is how flexible it is when future modifications on the real FMS are made. Unlike software-based simulation, simulation components in hardware-based parallel simulation have physical limitations which are very difficult to overcome. These physical limitations are restricted by the maximum board size of printed circuit boards, the number of microprocessors available on the board, the number of digital wires, etc. But limited flexibility in simulation modeling can be achieved by adopting reconfigurable design methodology. For example, wires can jump on the board in case there is a change on AGV's path network. Few redundant docking ports can be pre-allocated on the board when later on more machining centers might be added into the FMS. Modularized cards to simulate different types of working conditions can be pre-designed and plugged into those pre-left slots when basic operating pattern of the FMS is changed. Although limited flexibility can be provided in this way, for situations where a high flexibility requirement is necessary, this hardware-based parallel simulation mechanism is not very suitable.

CHAPTER 6

EXPERIMENTS AND DISCUSSIONS

Three experiments were designed to test the hardware-based parallel simulation on the prototype simulator. Details about these experiments are described first, including simulation scenarios for 1-workstation, 2-workstation, and 4-workstation manufacturing cells that simulated by using multiple DSPs on the simulator. Experimental results show that an average speedup of hardware-based parallel simulation to software-based simulation is at a range from 50 to 140. Slight variations can be added upon this speedup by a fused effect of various factors, such as system size, total events to be simulated, parallel processing sequence, and random number streams. Among them, three factors as the major driving forces to achieve this speedup are summarized as follows:

1. Simplification of simulation programming, because the specific hardware architecture for simulation is adapted.
2. Directly programming in assembly language, which takes full advantage of using special instructions provided by DSP technology.
3. Multi-microprocessor architecture allows a balanced distribution of computation intensive tasks among several microprocessors.

The experiments also reveal some limitations of such hardware-based parallel simulation in flexibility and ability to handle complex decision-making requirements.

6.1 General Procedure to Develop Experiments on the Simulator

The general procedure to develop a simulation experiment on the prototype simulator follows these four steps. First, a simulation scenario is designed. The task includes describing the simulation situation and specifying all the simulation parameters. Second, according to the simulation design, assembly programs are coded. Third, the assembly programs must be debugged on the board to ensure there are no errors in the program that may distort the simulation results. Lastly, simulation programs are executed on the board to generate simulation results to be collected and analyzed.

Figure 6.1 illustrates the general flow to develop and debug the code for the simulation on the board. The assembly programs are first edited using a text editor (WinEdit.exe) and saved as .asm files. Figure 6.2 shows a portion of such a source code. Then the assembler (dsk3a.exe) translates these assembly codes into machine language object codes for TMS320C31 DSP. Next, the machine language object code can be downloaded by the debugger to target DSPs and executed under the control of the debugger. If errors are found in the code, the source codes should be changed accordingly and the whole process repeated until no errors can be found in the program.

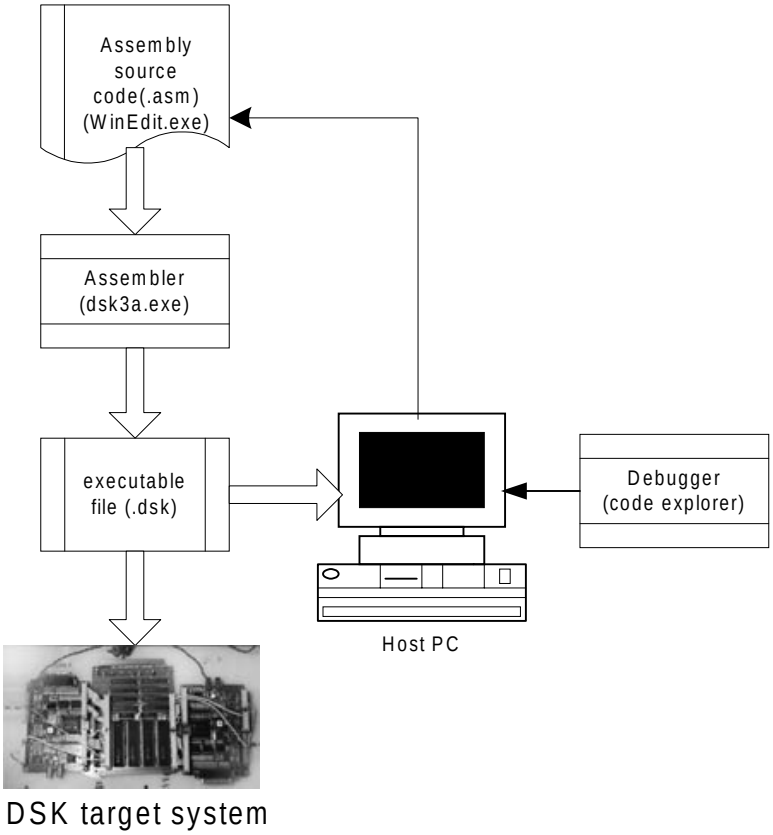


Figure 6.1 Development flow of DSP assembly program

```
*-------------------------------------------------------------------------
* 4MHardL1.ASM – Hardware-based parallel discrete event simulation
* of a scenerio: 4 machine, 2 part types, 2 robots, and considering machine breakdown
* This is the program running on DSP 1, used to simulating Lathe 1
* Overall PDES using distributed event list and distributed simulation clock
* Programmed by: Dong Xu
* Date         : March 22, 2001
* Modified     : Apr. 2, 2001
*-------------------------------------------------------------------------

    .sect "code"
    .entry  _main
*-------------------------------------------------------------------------
* FUNCTION DEF : _main
* Control the advance of the program
*-------------------------------------------------------------------------
_main:
        LDP     @_main           ;Set Data Segment Pointer
        LDI     @STACK,SP        ;Set stack pointer
        LDI     @PSharedMem2,AR5 ;The index of flags in shared memory section 2, for DSP 3
        LDI     @PSharedMem1,AR6 ;The index of flags in shared memory section 1, for DSP 2

        CALL    _initial         ;Initialize the simulation, especially data structure
        CALL    _barrier

;Setup Timer 0 to count the execution time
        LDI     @BISR, R0
        STI     R0, @9FC9h      ;Save the branch instruction to Timer0 interrupt vector
        LDI     200h, R1
        STI     R1, @8020h      ;Using internal H1/2 clock for Timer 0
        LDI     @PERIOD, R1
        STI     R1, @8028h      ;Set period of Timer 0
        LDI     0,R1
        STI     R1,@8024h       ;Reset counter register
        LDI     2C0h, R1
        STI     R1, @8020h      ;Start Timer 0
        OR      @ETINT0, IE     ;Enable Timer 0 interrupt
        OR      @EINT, ST       ;Enable Interrupt

LM1     LDF     @Time,R0
        CMPF    @EndTime,R0     ;Does simulation end?
        BGT     LM2

        CALL    _timing         ;Determin the next event type

;Invoke the appropriate event handle function
        LDI     @PSUB,AR0
        ADDI    @NxtEvet,AR0
        LDI     *AR0,R0
        CALLU   R0              ;Handle part arrival, departure, or machine breakdown events
         B       LM1
LM2
        CALL    _report         ;Report simulation results

        LDI     0,R0
        STI     R0,@SysExit     ;Successfully run
        ADDI    R1, R0
        STI     R0, @TOTCYC     ;Total cycle used
END     B       $               ;End of the program
```

Figure 6.2 Example DSP source code

Code Explorer, developed by GO-DSP Inc., is the debugger software used in this work. It is a visual development software running on the host PC for debugging DSK assembly codes. Figure 6.3 shows the user interface of the debugger. The advantage of using this simple debugger is that it only downloads a relative small kernel (about 256 bytes) into the on-chip memory, leaving more memory space for applications. But this debugger does not have the capability for parallel debugging, so one host PC is needed to control one DSK. To form a parallel developing system, four host PCs are linked to the simulator through parallel ports. Figure 6.4 shows a picture of this parallel developing environment.

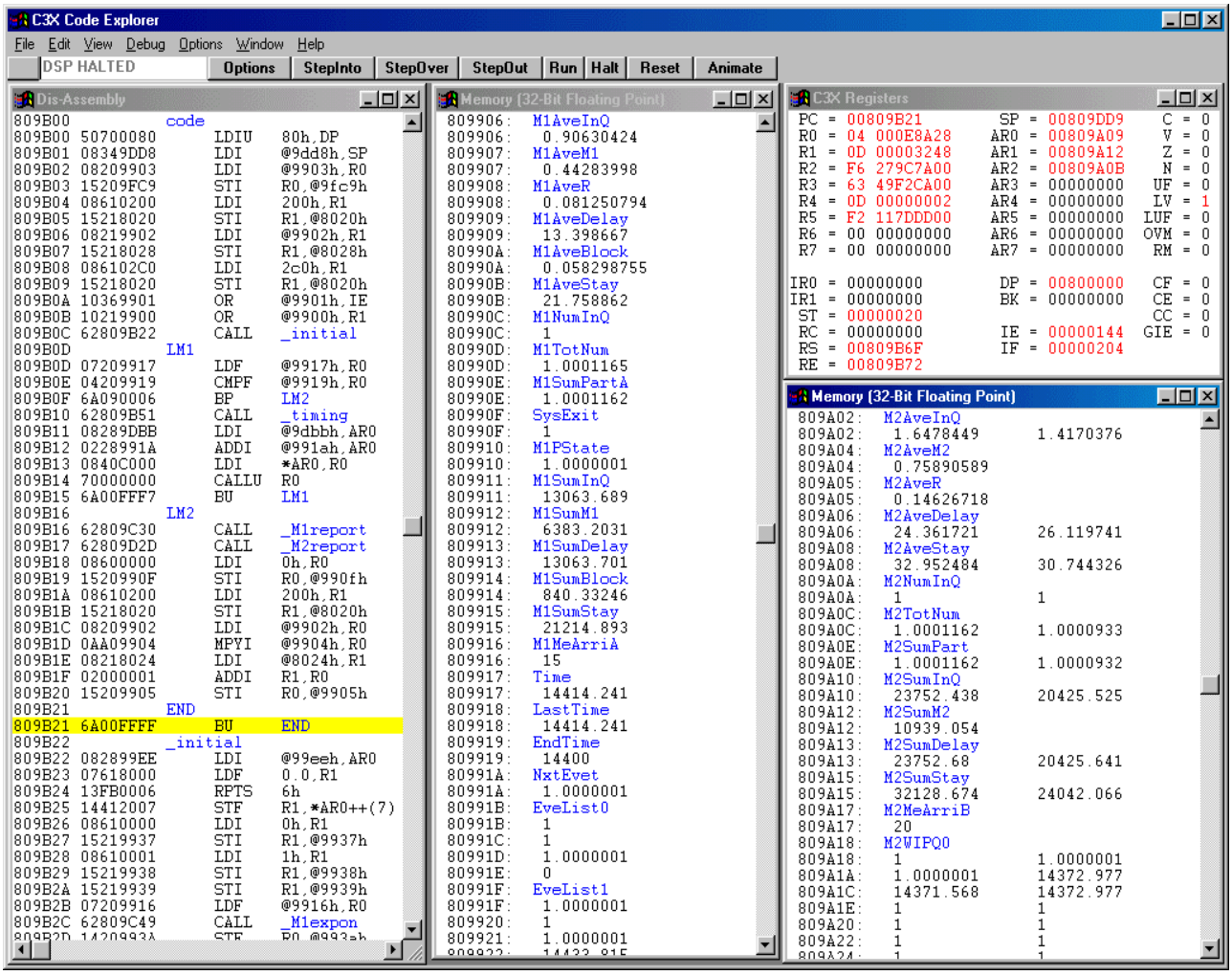


Figure 6.3 User interface of the debugger for DSKs

Figure 6.4 Parallel developing environment for the prototype simulator

The experiment design covers simulation cases for 1-workstation, 2-workstation, and 4-workstation manufacturing cells. In following sections, details about the experiment design and simulation execution are described. Simulation results for each experiment are collected manually and carefully analyzed later on.

6.2 Experiment Case for 1 Workstation

In this experiment, a one-workstation manufacturing cell is simulated using 2 DSPs. As shown in Figure 6.5, the cell contains 1 CNC lathe and one robot, and processes one type of part. The processing time is of exponential distribution with mean of 6.0 minutes. A conservative parallel discrete event simulation model and a modified hardware-based parallel simulation model were programmed on the prototype simulator for this scenario. In conservative PDES, one

DSP is allocated to simulate arrivals, and the other is for departures. Part-passing between these two DSPs is merely by checking flags and messages. In hardware-based parallel simulation, two distinguished features are implemented. First, part-passing is realized by combining a direct digital link and flag checking. Second, timing is controlled by timers with the time scaling simulation method. Serving as a reference, a software simulation model was built by using ProModel. One thing worth mentioning is that, because there is no way to control the internal execution mechanism in ProModel, the software simulation model cannot be built exactly the same as those programmed on the simulator. Efforts have been made to build the software simulation models as closely as possible to its hardware-based counterparts.

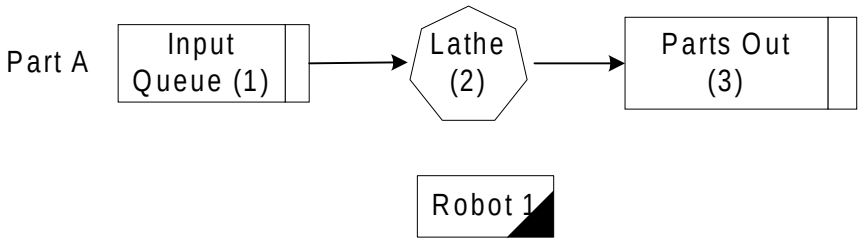


Figure 6.5 Simulation scenario of a one-workstation cell

During this experiment, 15 replications with different sets of seeds were executed for each method. Each replication simulated a continuous 240-hour working period. Same seed sets were used for the conservative PDES and the hardware-based parallel simulation. But the random streams used in software-based simulation model cannot be synchronized with the hardware-based parallel simulation model because random numbers for the hardware-based simulation are generated by intentionally chopping off a long digital series using the unique feature of hardware, which cannot be realized by a software subroutine on regular personal computers. Table 6.1 summarizes the average and standard deviation of interested parameters generated by all three methods. On average, software simulation finished one replication in 1.68 seconds on a Pentium II 350 PC, and conservative PDES finished one run in 0.035 seconds on the simulator, and the hardware-based parallel simulation at about 0.026 seconds. Hence, hardware-based parallel simulation is about 63.6 times faster than the software simulation, and 34.6% faster than the conservative PDES.

Table 6.1 Average and standard deviation of interested parameters for 1 workstation scenario

Interested Variables		**Software Simulation**	**Conservative PDES**	**Hardware PDES**
Execution Time (cycles)		**1.68 seconds**	**0.035 seconds (878551 cycles)**	**0.026 seconds (652634 cycles)**
Total Throughput Part A		967.5 (23.2)	970.1 (23.9)	970.1 (23.9)
Resource Utilization (%)	Lathe	40.2% (1.3%)	40.1% (1.3%)	40.1% (1.3%)
	Robot	8.2% (0.2%)	8.1% (0.2%)	8.1% (0.2%)
Average Number in Queue		0.413 (0.056)	0.316 (0.054)	0.316 (0.054)
Average Stay (min.)		12.876 (0.786)	11.247 (0.862)	11.247 (0.862)

One interesting point is to compare the real execution time with the estimated execution time. For this case, the computing difficult coefficient m is about 1.5, the scaling factor used is 10, the simulation time period is 240×3600 seconds, and the frequency of the digital clock is 50×10^6 Hz. With all these parameters, the estimated time for one replication $Ts = \frac{1.5\times240\times3600}{10\times50\times10^6} = 0.0259$ seconds, which is very close to the real average execution time 0.026 seconds.

Two sample t-test were performed on several interested parameters to validate the hardware-based parallel simulation by comparing their means to those generated by the software-based simulation. Table 6.2 lists all the data from 15 replications, in pairs for parameters like throughput, utilization of machine, utilization of robot, and total stay in system. Statistical analysis shows that at 95% confidence level, the means of throughput and utilization of machine are equal. For throughput, the 95% confidence interval is between (-15.1, 20.3). T value is 0.30, much less than the standard T value of 2.145 (Degree of freedom at 14). For utilization of machine, the 95% confidence interval is between (-0.98, 0.83). But utilization of robot and total stay in system are not equal. Both average values generated by the hardware-based parallel simulation are slightly less than those of software-based simulation. The reason is the simplification of the robot control in hardware-based parallel simulation. Such simplification helps to distribute the computation load of robot control among DSPs, while at the same time tries not reduce disturbance on final simulation results as less as possible. But, such simplification may cause unacceptable large disturbance to a large-scale FMS. For those large-

scale manufacturing systems, dedicated microprocessors must be added to the simulator to simulate all the transportation system.

Table 6.2 Detail data of 15 simulation replications

Run No.	Throughput		Utilization of Machine		Utilization of Robot		Stay in System	
	Hardware	Software	Hardware	Software	Hardware	Software	Hardware	Software
1	966	999	40.3	42.25	8	8.51	12.334	13.380
2	960	973	39.7	40.64	8	8.28	12.192	13.283
3	942	961	39.9	38.27	7.8	8.18	10.78	11.754
4	969	969	40.5	40.41	8.1	8.25	12.058	12.991
5	1010	954	40.4	40.43	8.4	8.12	10.707	12.266
6	985	982	39.2	40.89	8.2	8.36	10.076	14.156
7	916	938	38.8	39.83	7.6	7.99	10.786	13.168
8	979	993	41.6	42.06	8.1	8.46	12.694	12.349
9	958	963	41	40.99	8	8.2	11.188	12.912
10	952	1004	37.5	41.24	7.9	8.54	9.524	13.523
11	1002	982	41.9	39.68	8.3	8.36	10.923	12.841
12	975	941	39.6	37.68	8.1	8.01	10.495	11.260
13	967	925	39.6	39.83	8	7.87	11.716	12.927
14	973	948	40.6	38.91	8.1	8.07	11.399	12.325
15	998	981	41.5	40.13	8.3	8.35	11.832	13.99

Degree of freedom = 14 95% confidence level T value = 2.145

```
95% C.I. for throughput: ( -15.1,  20.3)
T-Test = (vs not =): T= 0.30  P=0.77

95% C.I. for utilization of machine: ( -0.98,  0.83)
T-Test = (vs not =): T= -0.17  P=0.86

95% C.I. for utilization of robot: ( -0.327,  -0.026)
T-Test = (vs not =): T= -2.41  P=0.023

95% C.I. for stay in system: ( -2.26,  -1.00)
T-Test = (vs not =): T= -5.31  P=0.0000
```

To further explore the speed issue, another set of experiments was designed. The idea is that when changing the mean time between arrivals, the total amount of events to be handled during the 240-hour simulation period can be changed, and in turn, the total computation time to finish the simulation shall change accordingly. In experiments, mean time between arrivals was changed in a decreasing order from 30 minutes to 7.5 minutes, with an increasing throughput

from 499 to 1896. Table 6.3 lists each value of mean time between arrivals used in the experiment and corresponding throughput. It also lists the execution time of all three models, and speedup of the hardware-based parallel simulation to the other two. On average, speedup of hardware-based parallel simulation to software simulation is about 54.3, and to conservative PDES is about 1.28. Figure 6.6 shows the trend of all three execution times changing with different throughputs.

Table 6.3 Execution times vs. throughput for all three models (1-workstation case)

MTBA	No. Part	Software (seconds)	PDES (seconds)	HARD (seconds)	Speedup 1	Speedup 2
30	499	0.8637	0.0208	0.0164	52.569	1.266
25	571	1.0330	0.0228	0.0178	58.016	1.281
20	708	1.2663	0.0269	0.0207	61.200	1.300
15	964	1.6300	0.0354	0.0269	60.634	1.315
12.5	1132	1.9400	0.0413	0.0313	61.920	1.318
10	1405	2.3830	0.0527	0.0403	59.180	1.308
8.5	1694	2.7253	0.0691	0.0541	50.378	1.277
8.0	1794	2.8797	0.0774	0.0615	46.831	1.259
7.5	1896	2.9950	0.0953	0.0784	38.210	1.216
Average					**54.327**	**1.282**

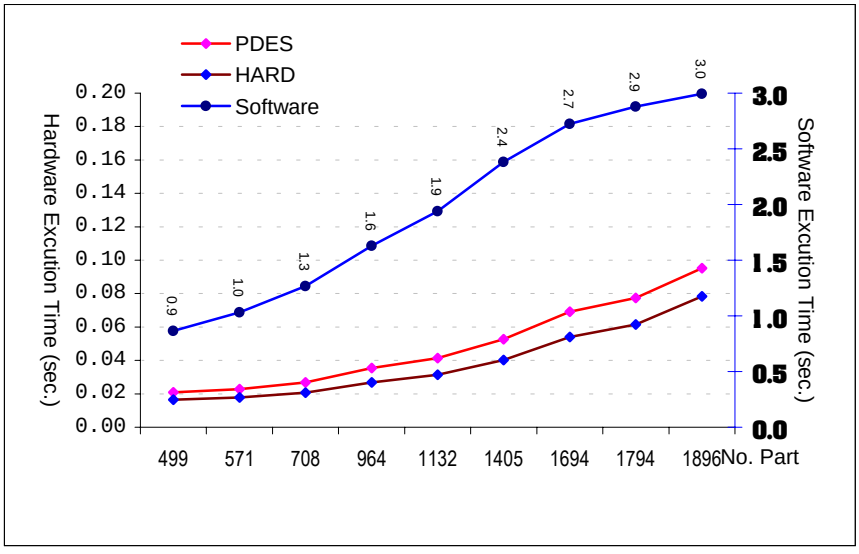


Figure 6.6 Plots of execution times vs. throughput (1-workstation case)

6.3 Experiment Case for a 2-Workstation Production Line

In this experiment, a 2-workstation production line, shown in Figure 6.7, is to be simulated by using 2 DSPs. The cell is composed of one CNC lathe and one CNC Mill, two robots (serving each workstation), and several buffers (queues). Two types of parts are processed in the system. Part A needs operation on both machines, with processing time of Exp(6.0) on Lathe and Exp(8.0) on Mill, respectively. Part B only goes through the Mill, with an processing time of Exp(4.0).

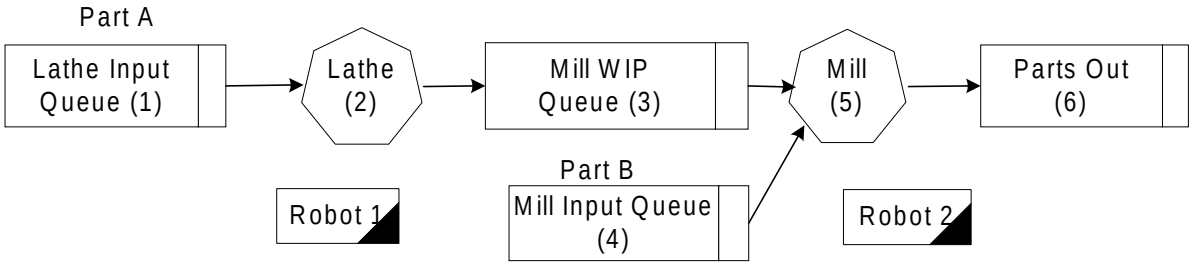


Figure 6.7 Simulation scenario of a 2-workstation production line

The schedule for each part type is as follows. Part A first arrives at the input queue of Lathe, and waits until robot 1 transports it to the Lathe. After finishing its operation on the Lathe, part A is moved by robot 1 to the working-in-process storage and waits there. Later, Robot 2 picks up part A to the Mill. After milling operation, part A is transported by robot 2 out of the system. Part B arrives at the input queue of the Mill first, and then picked up by robot 2 to the Mill. When part B is done on the Mill, robot 2 moved it out of the system. There is a routing decision for robot 2 to choose which part from the two queues in front of Mill to process. In this experiment, the rule of arrival time was used. The rule was implemented to compare the very first waiting part between these two queues, and whichever has the smaller entering time will be chosen as the next part.

The capacity of the working-in-process queue is 5. When this storage is full and there is more part A coming from the Lathe, the Lathe shall be blocked until there is a space after the Mill finishes its current operation and the next part is picked from this working-in-process queue.

Moreover, the Lathe might be broken. The broken pattern is defined by two parameters: mean time between failure and mean time to repair. Because a stochastic broken pattern does not bring any benefit to the main focus of this research, fixed values were used. To be specific, the mean time between failures for Lathe is 450 minutes, and the mean time to repair is 25 minutes.

Two DSPs were used for both conservative PDES and the hardware-based parallel simulation in this case. One DSP simulated the activities around the arrival of part A, Lathe, and robot 1, and the other one simulated the activities around arrival of part B, Mill, and robot 2. The same timing and part passing mechanism for conservative PDES and hardware-based parallel simulation were used here as in simulating the 1-workstation case. A software simulation model was also built as closely as possible to the models programmed on the simulator.

During this experiment, 15 replications with different sets of seeds were executed for each model. Each replication simulated a continuous 240-hour working period. Same seed sets were used for conservative PDES and the hardware-based parallel simulation, but not the software simulation model. Table 6.4 summarizes the average and standard deviation of interested parameters generated by all three models. For the average execution time, software simulation finished one replication in 3.53 seconds on a Pentium II 350 PC, while conservative PDES finished one run in 0.052 seconds on the simulator, and hardware-based parallel simulation at about 0.047 seconds. On average, hardware-based parallel simulation is about 75.7 times faster than the software simulation, and 17.3% faster than the conservative PDES.

When comparing the real execution time with the estimated execution time, for this case, the computing difficult coefficient m is about 2.5, the scaling factor used is 10, the simulation time period is 240×3600 seconds, and the frequency of the digital clock is 50×10^6 Hz. With all these parameters, the estimated time for one replication $Ts = \frac{2.5\times240\times3600}{10\times50\times10^6} = 0.0432$ seconds, which is also very close to the real average execution time 0.047 seconds.

Table 6.4 Average and standard deviation of interested parameters for a 2-workstation scenario

Interested Variables		**Software Simulation**	**Conservative PDES**	**Hardware PDES**
Execution Time (cycles)		**3.53 seconds**	**0.054 second (1,296,128)**	**0.046 second (1,175,946)**
Total Throughput	Part A	941.4 (28.3)	965.8 (23.6)	965.8 (23.6)
	Part B	719.7 (24.5)	721.9 (31.47)	721.9 (31.47)
Resource Utilization (%)	Lathe 1	39.5% (1.0%)	40.9% (2.1%)	40.9% (2.1%)
	Mill 1	71.8% (1.0%)	73.7% (3.2%)	73.7% (3.2%)
	Robot 1	10.1% (0.3%)	8.1% (0.2%)	8.1% (0.2%)
	Robot 2	19.2% (0.4%)	15.5% (0.3%)	15.5% (0.3%)
Average Number in Queue	Lathe 1 Input Queue	0.825 (0.066)	0.938 (0.425)	0.938 (0.425)
	Mill 1 Input Queue	1.140 (0.082)	1.307 (0.341)	1.307 (0.341)
	Mill 1 WIP Queue	2.352 (0.161)	1.617 (0.355)	1.617 (0.355)
Average Stays in the System (min.)	Part A	67.717 (4.004)	54.489 (5.245)	54.489 (5.245)
	Part B	29.805 (1.394)	30.429 (6.580)	30.429 (6.580)

Experiments to manipulate the mean time between arrivals were conducted as well in this case. In experiments, mean time between arrivals for both part A and part B were changed within a range from 15 minutes to 25 minutes. Totally 9 pairs of such arrival patterns were tested. Table 6.5 lists specific pairs of mean time between arrivals used. For example, (20,15) means the mean time between arrivals for part A is 20 minutes, and for part B is 15 minutes. The table also lists the execution time of all three models, and speedup of the hardware-based parallel simulation to the other two. On average, speedup of the hardware-based parallel simulation to software simulation is about 83.1, and to conservative PDES is about 1.12. Figure 6.8 shows the trend of all three execution times with respect to these arrival patterns.

Table 6.5 Execution times of different arrival pattern for all three models (2-workstation case)

Arrival Pattern	Software (seconds)	PDES (seconds)	HARD (seconds)	Speedup 1	Speedup 2
(25, 25)	2.6723	0.0310	0.0297	89.991	1.044
(25, 20)	2.9840	0.0354	0.0333	89.689	1.065
(25, 15)	3.5140	0.0433	0.0401	87.581	1.080
(20, 25)	2.8617	0.0392	0.0335	85.421	1.171
(20, 20)	3.2480	0.0436	0.0370	87.867	1.179
(20, 15)	3.7030	0.0519	0.0436	84.880	1.189
(15, 25)	3.2327	0.0475	0.0441	73.335	1.077
(15, 20)	3.5317	0.0537	0.0458	77.054	1.171
(15, 15)	3.9210	0.0636	0.0542	72.278	1.172
Average				**83.122**	**1.128**

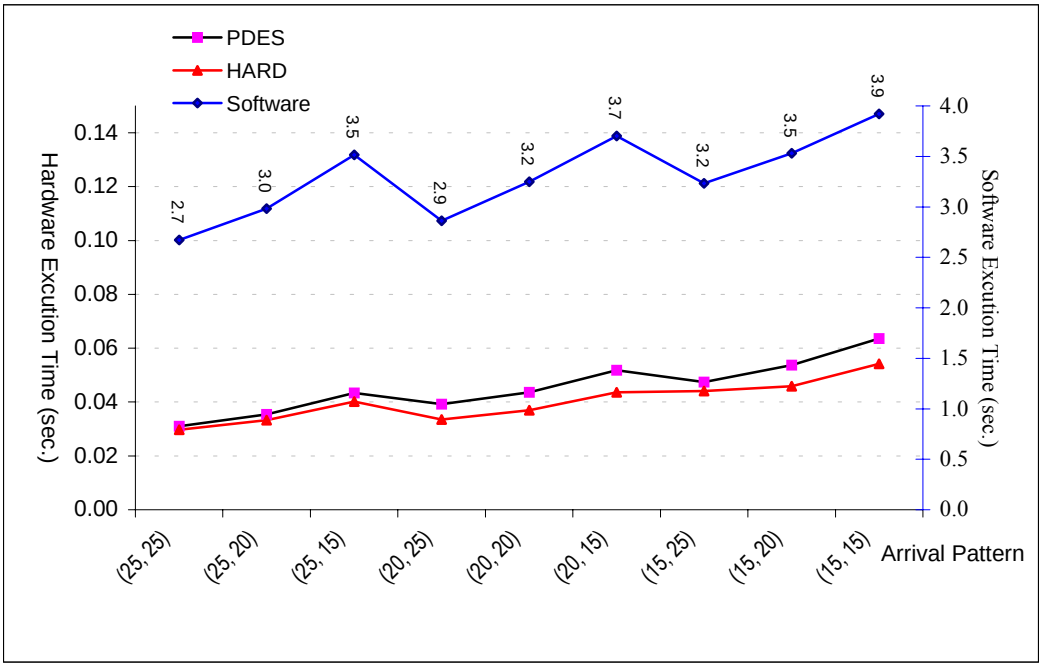


Figure 6.8 Plots of execution times vs. arrival patterns (2-workstation case)

6.4 Experiment Case for a 4-Workstation Manufacturing Cell

In this experiment, a 4-workstation manufacturing cell, shown in Figure 6.9, is simulated. This simulation scenario comes from the laboratory FMS, and can also be viewed as putting 2 sets of previous 2-workstation manufacturing cell together. Actually, most parameters are the same as the 2-workstation cell, except there is a new routing path between Lathe 1 and Lathe 2. This new path is used to route part A from Lathe 1 to Lathe 2 during the broken period of Lathe 1. If Lathe 1 is broken, the very first part A waiting in Lathe 1's input queue will be routed to the input queue of Lathe 2, and continue its operation from Lathe 2 to Mill 2.

The detail of this routing procedure is described as follows. At the beginning of the broken period, robot 1 transports the first part waiting in Lathe 1's input queue to a temporary storage (index table) between Lathe 1 and Lathe 2. This part will be picked up by robot 2 some time later and inserted into Lathe 2's input queue. In which position the part should be inserted is determined by the rule used for the queue. In this experiment, the dispatching rule is AT (Arrival Time). Therefore, by comparing the entering time of this routed part to those parts already waiting in the queue, the correct position of this part in the queue can be found. Later, after Lathe 2 processes this routed part, another part is allowed to be routed from Lathe 1. This routing process goes on until Lathe 1 is repaired and returns back to work.

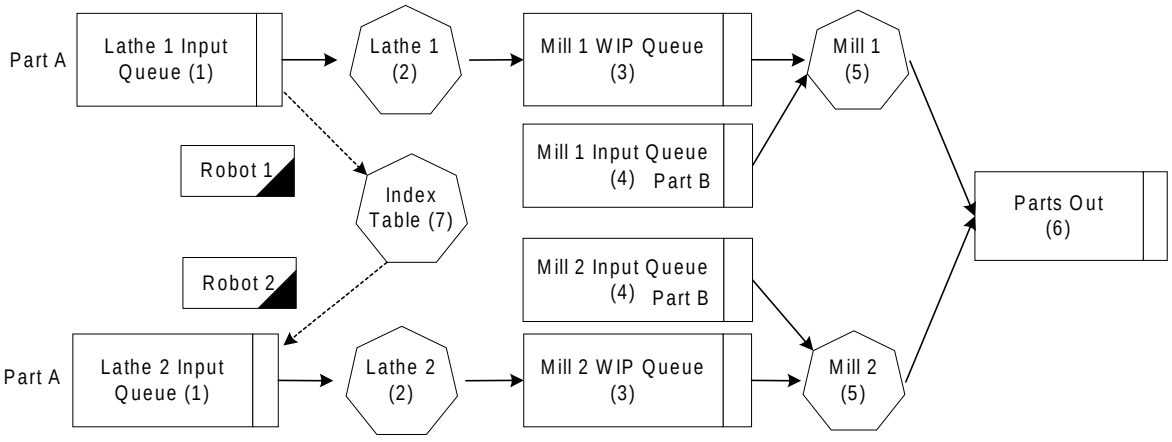


Figure 6.9 Simulation scenario of a 4-workstation manufacturing cell

Because there are only 2 timers on each DSP used for the prototype simulator, the hardware-based parallel simulation cannot be performed for this 4-workstation manufacturing cell. Rather, two conservative PDES models using two different submodeling methods with the support of direct digital links were programmed. One submodeling method makes robots as the core objects and the other one makes workstations as the core objects. For the first submodeling method, 2 DSPs are used to simulate the two robots in the system. One DSP is dedicated to simulate all activities around robot 1, plus arrivals and operating on Lathe 1 and Mill 1. The other DSP is to simulate all activities around robot 2, plus arrivals and operating on Lathe 2 and Mill 2. For the submodeling method focusing on workstations, 4 DSPs are used, and each is mapped to one workstation. Specifically, DSP 1 simulates activities around Lathe 1, such as part A arrival on Lathe 1, routing part A from Lathe 1 to Lathe 2, passing part A from Lathe 1 to Mill 1, maintaining the input queue and partial control of robot 1. Similarly, DSP 2 simulates all activities around Mill 1 and partial control of robot 1, DSP 3 for Lathe 2 and robot 2, and DSP 4 for Mill 2 and robot 2. A software simulation model was also built as closely as possible to the 4 DSP simulation model programmed on the simulator.

Same as the other two experiments, 15 replications with different sets of seeds were executed for each model in this experiment. Each replication simulated a continuous 240-hour working period. Same seed sets were used for 2 DSP and 4 DSP simulation case, but not the software simulation model. Table 6.6 summarizes the average and standard deviation of interested parameters generated by all three models.

For the average execution time, software simulation finished one replication in 8.404 seconds on a Pentium II 350 PC, while conservative PDES of 2 DSP finished one run in 0.107 seconds on the simulator, and 4 DSP model at about 0.060 seconds. On average, 4 DSP model is about 139.1 times faster than the software simulation, and 78.3% faster than the 2 DSP conservative PDES. When the computation efficiency is considered, the efficiency by adding two more DSPs is 1.78/2 = 0.89. Considering the fact that the “safe” time synchronization method can only lead to more waiting and waste more computing time to avoid deadlock when more microprocessors are used, this speedup gives a good estimation if the hardware-based parallel simulation is applicable.

Table 6.6 Average and standard deviation of interested parameters for a 4-workstation scenario

Interested Variables		**Software Simulation**	**Conservative PDES (2 DSP)**	**Conservative PDES (4 DSP)**
Execution Time (cycles)		**8.404 seconds**	**0.107 seconds (2677589)**	**0.060 seconds (1505416)**
Total Throughput	Part A	1899.9 (41.0)	1904.4 (59.3)	1910.3 (65.6)
	Part B	1444.5 (34.7)	1437.8 (62.0)	1439.5 (35.1)
Total Routed	Part A	39.3 (8.8)	17.5 (5.2)	20.3 (6.8)
Resource Utilization (%)	Lathe 1	38.0% (1.6%)	38.6% (2.1%)	39.5% (1.7%)
	Mill 1	70.6% (1.4%)	71.3% (2.7%)	71.2% (2.4%)
	Lathe 2	41.2% (1.3%)	40.4% (2.4%)	39.9% (2.1%)
	Mill 2	75.5% (2.2%)	73.6% (2.3%)	74.0% (2.7%)
	Robot 1	29.4% (0.4%)	38.8% (1.4%)	37.5% (0.9%)
	Robot 2	34.7% (0.4%)	40.9% (1.7%)	38.9% (1.4%)
Average Number in Queue	Lathe 1 Input Queue	0.667 (0.275)	0.980 (0.384)	0.947 (0.269)
	Mill 1 Input Queue	1.091 (0.116)	1.372 (0.216)	1.350 (0.196)
	Mill 1 WIP Queue	1.529 (0.142)	1.718 (0.229)	1.724 (0.275)
	Lathe 2 Input Queue	1.189 (0.473)	1.171 (0.624)	1.118 (0.398)
	Mill 2 Input Queue	1.368 (0.328)	1.484 (0.415)	1.603 (0.251)
	Mill 2 WIP Queue	2.063 (0.301)	2.002 (0.340)	2.011 (0.316)
Average Stays in the System (min.)	Part A	64.538 (10.559)	65.173 (12.027)	63.885 (10.132)
	Part B	31.532 (5.689)	35.791 (5.020)	36.620 (5.116)

Experiments to manipulate the mean time between arrivals were conducted as well in this case. In experiments, mean time between arrivals for both part A and part B were changed within the range [15, 25]. Totally 9 pairs of such arrival patterns were tested. Table 6.7 lists each specific pair of mean time between arrivals used. For example, (20,15) means the mean time between arrivals for part A is 20 minutes, and for part B is 15 minutes. The table also lists the execution time of all three models, and the speedup of 4-DSP to the other two. On average, speedup of using 4 DSP to the software simulation model is about 136.1, and to 2-DSP PDES is about 1.62. Figure 6.10 shows the trend of changing execution times.

Table 6.7 Execution times of different arrival pattern for all three models (4-workstation case)

Arrival Pattern	Software (seconds)	2DSP (seconds)	4DSP (seconds)	Speedup 1	Speedup 2
(25,25)	5.5010	0.0613	0.0355	155.105	1.727
(25,20)	6.1000	0.0669	0.0390	156.546	1.716
(25,15)	6.9100	0.0800	0.0475	145.379	1.683
(20,25)	6.3300	0.0715	0.0485	130.484	1.474
(20,20)	6.8143	0.0776	0.0525	129.741	1.477
(20,15)	7.7400	0.0940	0.0610	126.942	1.542
(15,25)	7.8813	0.0983	0.0592	133.238	1.661
(15,20)	8.4047	0.1071	0.0654	128.516	1.637
(15,15)	8.8793	0.1240	0.0746	119.038	1.663
Average				**136.110**	**1.620**

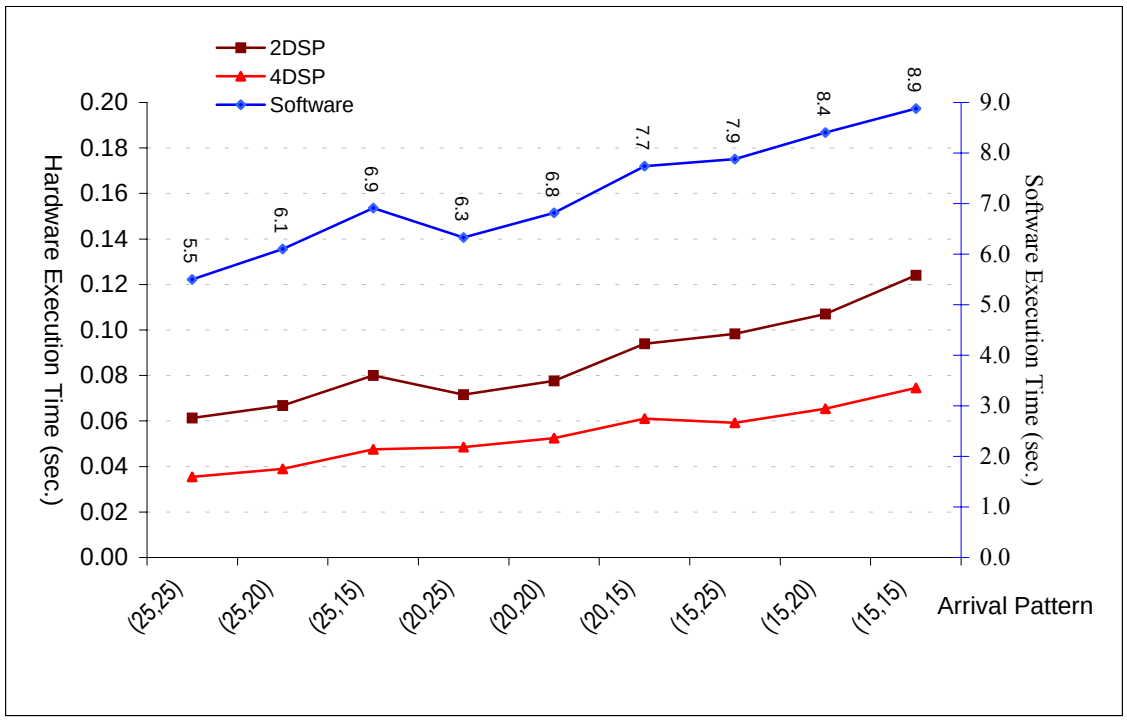


Figure 6.10 Plots of execution times vs. arrival patterns (4-workstation case)

6.5 Discussions

In summary of above experiments, the speedup of the hardware-based parallel simulation to software simulation is at a range between 50 and 140, and the speedup to conservative PDES is at a range from 15% to 30%. This result proves that the hardware-based parallel simulation is promising in dramatically reducing the computational time, and performs better than conservative PDES. Figure 6.11 summarizes this trend of speedup. Basically, the figure shows when the system size is small, speedup is also relatively small. As the system size increases, speedup increases accordingly. However, this increment can quickly reach its upper limit because of the physical limitations of such hardware-based simulator.

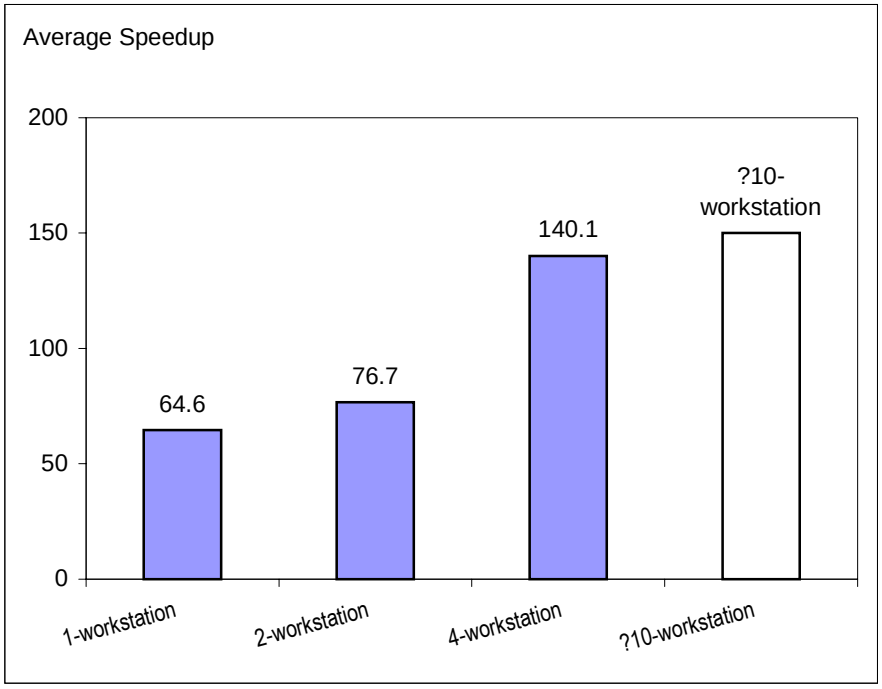


Figure 6.11 Average speedup of hardware-based parallel simulation to software simulation

The speedup shown in Figure 6.11 is achieved by an accumulated effect from various factors in the hardware-based parallel simulation. Three major features which contribute most to

this speedup include simplification of simulation, directly programming in assembly language, and specific multi-microprocessor architecture of the simulator. Among them, simplification of simulation plays the most important role in reducing the execution time. Such simplification is enabled by the multi-microprocessor architecture designed specifically for simulation purpose, which prevents losing too much accuracy in a statistical sense for such a simplification. Moreover, with the help of direct digital links between DSPs, simplification of timing control helps to avoid wasteful waiting and dangerous deadlock. Directly programming in assembly language enables more efficient codes by using special instructions of DSP technology, such as block repeat, delayed branches, and bytewise addressing. The multi-microprocessors architecture distributes some computation intensive tasks, such as random number generation and robot control, evenly into several microprocessors so as to gain more speedup.

In addition to above major features, some minor factors can also bring in variations to the speedup. These minor factors include system size, total number of events to be handled, and random number generating sequence. It looks like as the system size goes up, the speedup also goes up, but the efficiency of adding more DSPs really goes down. The effect is not consistent when considering how total number of events and different random number generating sequence may affect the speedup. These two factors can either increase or reduce the speedup because they cause different sequences of synchronization among DSPs, which in turn can either add or reduce some waiting cycles. But these minor factors definitely do not have the same level of power as major features to affect the speedup.

The improvement of hardware-based parallel simulation to conservative parallel simulation is mainly gained from three new technical features used in this work, namely, the DSP technology, the direct digital links, and the time scaling mechanism. The parallel computing capability of DSPs, specifically, the interlock instructions, provides a more applicable parallel environment for this work. Direct digital links help to simplify the timing mechanism and reduce waiting cycles for parallel simulation. It proves that the ability to manipulate at board level can help in improving the simulation, while no changes can be applied if buying an expensive general parallel computing machine. The time scaling method, implemented by using timers in this work, is a valuable small step towards a true hardware-based parallel simulation, where a massive parallel computing power can realize the time scaling simulation method to achieve more speedup.

Another discussion about the developing difficulty are also conducted through these experiments. Developing difficulty for both software-based simulation method and hardware-based parallel simulation method will increase when system size increases. There are not many reusable components for the software simulation model, which means for different simulation scenario, different model has to be built. On the contrary, once the simulation program is coded on one DSP, most of its subroutines are reusable to other DSP, and to other simulation scenario. In this way, the increasing of the developing difficulty by using the hardware-based parallel simulation can be keep proportional with respect to the increasing of the system size. For example, in the 4-worksation case, there are total around 3000 lines of code programmed on 4 DSPs, and for the 1-workstation case, there are total around 1200 lines of code programmed on 2 DSPs. The increase of code versus the increase of the microprocessor used is only at a ration of 1.25. Moreover, most part of the code is reusable to other DSPs, which also significantly reduce the developing difficulty of the hardware-based parallel simulation method.

It is not surprising to reveal many problems with the hardware-based parallel simulation method as this research tries to break into a new field. Basically, there are two major problems which could restrict the application of hardware-based parallel simulation. One problem is concerning the limited flexibility of the simulator, and the other one is related to handling the computation intensive decision-making tasks in the simulation. Once the circuit board is designed and soldered, all components, such as microprocessors, timers, direct digital links, are fixed, which enforce a physical limitation at the modeling flexibility on the simulator. For example, the number of microprocessors planted on the board restricts the maximum number of workstations that can be simulated in the FMS, and the number of external timers limits the types of events that can be handled in the timing subroutines, and the number of digital wires decides the maximum number of part types that can be simulated in the system. Generally speaking, larger changes on simulated FMS, such as rearranging the whole layout of a shop floor, adding a larger number of new equipment, adding too many new part types, can lead to obsolescence of the old simulator and a requirement to develop a new simulator. Slight changes, such as adding or removing a few pieces of equipment (workstations, robots, pallets, temporary storages, minor change on AGV's path network, etc.), can be adapted by some sort of reconfigurable design. For example, wires can jump on the board to represent a change on robot's path network. One microprocessor can be disconnected from the board to represent removing a workstation from

the system. Several redundant docking ports can be pre-designed on the board for future extension of the system. Of course, comparing to almost unlimited flexibility of software simulation, such reconfigurable design has its limitation and only works for minor changes.

The other problem shows that the hardware-based parallel simulation cannot provide a superior advantage when computation intensive tasks are involved. Such computation intensive tasks usually happen when complex decision-making processes or algorithms are implemented within simulation. In most cases, it is involved with the implementation of dispatching rules. Simple dispatching rules, such as FIFO (first in first out), AT (arrival time), SPT (shortest process time), EDD (earliest due date), LAU (lowest average utilization), which require only one-step ahead information from other microprocessors, can be easily implemented. Dynamic dispatch rules, which employ fuzzy selection, intelligent control (neural network or expert systems), or optimization on selecting dispatching rules, do require an intensive computation at each decision-making point. Because there is no way to simplify such computations in the hardware-based parallel simulation, it cannot achieve a large speedup under these situations, which means the hardware-based parallel simulation cannot provide a superior advantage to software simulation when simulating such systems.

CHAPTER 7

CONCLUSIONS AND FUTURE RESEARCH

7.1 Conclusions

This research presents a new way to achieve the real-time simulation for FMS by using hardware technology directly for the simulation. The hardware-based parallel simulation refers to perform real-time simulation of FMS on a multi-microprocessor integrated digital circuit board specifically for the purpose of FMS simulation. A prototype simulator with architecture combining shared-memory and parallel I/O port was built. Experimental results show that the hardware-based parallel simulation mechanism has potential to achieve a dramatic improvement in speed compared to software-based simulation methods and is very promising in providing a cost efficient solution for on-line rapid simulation and control on shop floors. The method also has to overcome some serious limitations in modeling flexibility, which may restrict its applications.

Possible architectures for the construction of such a simulator are also explored using a mapping technology. These architectures are designed mainly based on the layout of the laboratory FMS and distinguished from each other by the way that communications are established among microprocessors. The bus-based architecture depends on the data bus to support the communication among the microprocessors. Multi-port SRAM is the key technology in the shared-memory-based architecture and the coordination will be achieved by accessing to a reserved area in the shared memory. For the parallel I/O port based architecture, messages are exchanged through communication ports. Comparison of these three architectures shows that there are advantages and disadvantages associated with each. The bus-based architecture is more suitable for large-scale FMS because it is relatively easy to expand, but at a higher cost. The shared-memory-based architecture has the expected computing power but its fixed printed circuit board (PCB) design makes the future expansion a relatively difficult task, and is therefore suitable only for small-scale FMS. This is also true for the parallel I/O port based architecture, although it is the easiest to be realized.

For the timing mechanism, an event-driven simulation making use of the time scaling technology is employed. The advantage of the time scaling technology is that by using the on-board digital clock, the sequence of simulation events are naturally sorted and is consistent with the real world. In this way, the synchronization overhead to maintain causality can be largely reduced. Moreover, the timing mechanism is simplified by combining with the unique hardware features on the board of the FMS specific simulator. This is the main reason that the hardware-based parallel simulation is a worthwhile endeavor.

7.2 Contributions

The most important contribution of this research is the introduction of a new idea about directly applying hardware components to facilitate a parallel simulation of manufacturing systems. Such a new concept is not commonly found in simulating manufacturing systems based on the literature survey as part of this research. Other contributions in theory and technology issues can be summarized as follows:

1. A time scaling simulation method was proposed for a hardware-based simulation to dramatically speedup the execution of simulation.
2. Actually building a parallel computing board specifically designed for the purpose of manufacturing systems simulation by using DSP technology.
3. Time scaling was tested and analyzed on such a prototype simulator as the very first step towards realization of true hardware-based simulation.

Overall, as the very first step, this research explores some new concepts about hardware-based simulation, and provides a basic parallel computing platform which can stimulate a continuous research on hardware-based simulation.

7.3 Future Research Directions

Some valuable future research directions initiated by this research effort are to explore further hardware oriented methods. The prototype simulator is really partially hardware oriented

and partially software oriented method because most of the simulation mechanism is still realized by software codes, in assembly language. However, more customized components can be designed specifically for simulation purpose so as to implement the simulator using more hardware technologies. Currently, no single microchip designed specifically for the purpose of a simulator is available. From this research, two customized microchip designs are proposed as future research directions. One is a microchip for random number generator and the other is a smart queue used to simulate queues in the simulator.

1. Random Number Generator

The random number generator is a microchip that can generate a random number for random variables according to their distribution. In simulation models with random variables, generating exactly desired distributions for these variables is critical to ensure the accuracy of the simulation results. Unfortunately, it costs a large portion of valuable computation time to generate these pseudo random numbers when executing a simulation process. The results from software-based simulation show that 8.7% of the total execution time is used to generate random numbers. If other portions of the code are simplified, this percentage for random generation can only increase. For a simple queuing network with 2 servers, the time needed to generate the random numbers can increase up to 23% of the total execution time. This fact holds because during one run of a simulation, there are tens of thousands of random numbers generated. Although the computational time for one number is negligible, after accumulating all these computations together, the total computational time needed to generate all random numbers is considerable. Thus, if a new way to reduce the time spent to generate random numbers is found, the total simulation execution can be further reduced. One way is to search for faster algorithms to generate random numbers. The other direction is to realize the random number generation by hardware so as to release the computation burden from main CPU. With the dramatic advancement of electronic technologies, we are able to design a customized microchip whose only purpose is to generate random numbers for simulations. Such a microchip is called a hardware random number generator. To support simulations, it must be capable of providing pseudo random sequence for various kinds of distributions.

Existing hardware random number generators on the market are all built for the cryptograph usage. Hence, they focus on generating true random numbers by making use of

technologies such as thermal noise, radio active decay and free-running oscillators. Their random number sequence cannot be regenerated. But for the purpose of simulation, to regenerate the random number sequence is important in order to repeat the simulation procedure. More importantly, existing random number generators cannot provide random numbers with respect to a specific distribution. The proposed random number generator is distinguished from other generators by focusing on simulation usage, which means it can provide a variety of random distributions with the capability to regenerate the same random number sequence.

As imagined, the microchip should have a core of digital signal processors with on-chip FPGA, SRAM and other peripheral components for random generation algorithms, and communication purpose. The DSP core serves as the main computing power for the chip. It can be programmed to realize some complex algorithm to generate some special kinds of distributions. FPGA can be programmed and reprogrammed for simple random number generating methods so as to further improve the speed. ROM is used to store some fixed distribution table. E^2PROM can be used to store historical sample data. RAM is the buffer to store pre-generated random numbers. Communication ports are embedded to establish outside communication links with other microchips on the board.

In theory, all available random number generation methods can be implemented by the electronic components on the chip. Linear congruential generator (LCG) and Tausworthe generator are the two basic uniform random number generation methods. LCG can be easily implemented on the DSP by using shifting operation while Tausworth method can be realized by FPGA to form a shift register chain. Besides the basic uniform random number generator, other methods can be divided into four categories: inverse transform, table-lookup, estimated algorithms and empirical distribution techniques. Inverse transform techniques use a uniform random sequence as its base. The inverse of the specified cumulative distribution function $R^{-1}(x)$ is used to generate the required distribution. With the known function $R^{-1}(x)$, the distribution can be easily programmed on the DSP. Table-lookup techniques can be used for all the discrete distribution and some continuous distributions without explicit formula, such as Student t distribution. The table can be stored in a ROM, and corresponding random numbers can be found from the table. Estimated algorithms use some formula or algorithms to generate a distribution from other distributions. These formula and algorithms can be programmed on DSP or FPGA. The final method is to gather some historical samples, store them in an E^2PROM and use

interpolation to generate the random number between those empirical points. Overall, almost all of the random number generating techniques can be implemented on the proposed microchip.

2. Smart Queue

Smart queue refers to a microchip that can be used to simulate temporary storages in the FMS (such as conveyors, working-in-process storage, and index tables). The unique feature of this microchip is that common dispatching rules can be programmed as firmware in the chip, and the microchip can automatically maintain its queues controlled by the firmware realizing different dispatching rules. This microchip can be used on the board to replace the logic queues programmed in the simulation. As revealed by this research, the hardware-based parallel simulation method does not have any advantage when complex decision-making tasks are involved in simulation. This idea of a smart queue could be a possible solution to this problem. Basically, this microchip is designed to maintain queues according to various dispatching rules to be simulated. The chip should have a CPU, some memory storages, and communication ports. The memory storage is the body of the queue. Next out-going part can automatically be picked up by the smart queue itself so as to release the computational burden of implementing such complex dispatching rules from the main DSP.

With the help of these special designed microchips, the simulator can be realized in a further hardware-oriented way, and the speedup of the hardware-based parallel simulation to the software-based simulation can be improved.

REFERENCE

[1] Alvarez-Vargas, R., Dallery, Y., and David, R., A Study of The Continuous Flow Model of Production Lines With Unreliable Machines and Finite Buffers, *Journal of Manufacturing Systems*, Vol. 13, No. 3, 1994, 221-234.

[2] Angelva, J., Piltonen, P., Distributed Real-Time Data Management in a Flexible Manufacturing System (FMS), *Advances in Manufacturing Systems*, 1994, 81-85.

[3] Baid, N. K., Nagarur, N. N., An Integrated Decision Support System for FMS: Using Intelligent Simulation, *International Journal of Production Research,* Vol. 32, No. 4, Apr., 1994, 951-965.

[4] Balsamo, M. S., Manconi, C. B., Rollback Overhead Reduction Methods for Time Warp Distributed Simulation, *Simulation Practice and Theory*, Vol. 6, No. 8, 1998, 689-702.

[5] Banks, J., Carson, J. S., Nelson, B. L., Discrete-Event System Simulation, Prentice Hall, New Jersey, 1996, ISBN 0-13-217449-9.

[6] Banks, J., (editor), Handbook of Simulation: Principles, Methodology, Advances, Applications and Practice, *John Wiley & Sons*, Inc, New York, 1998, ISBN 0-471-13403-1.

[7] Benson, D., Simulation Modeling and Optimization Using Promodel, *Proceedings of the 1997 Winter Simulation Conference*, 1997, 587-593.

[8] Bhattacharyya, S. S., Sriram, S., and Lee, E. A., Optimizing Synchronization in Multiprocessor DSP Systems, *IEEE Transaction on Signal Processing*, Vol. 45, No. 6, June 1997, 1605-1618.

[9] Bhuskute, H. C., Mize, J. H., Parallel Discrete Event Simulation of Manufacturing Systems, *Proceedings of the Industrial Engineering Research Conference*, 1993, 705-709.

[10] Bhuskute, H., Application of Parallel Processing for Object Oriented Discrete Event Simulation of Manufacturing Systems, Dissertation of Oklahoma State University, 1993.

[11] Caprihan, R., Wadhwa, S., Impact of Routing Flexibility on the Performance of An FMS-A Simulation Study, *The international journal of flexible manufacturing systems*, Vol. 9, 1997, 273-298.

[12] Chandy, K. M., Misra, J., Distributed Simulation: A Case Study in Design and Verification of Distributed Programs, *IEEE Transactions on Software Engineering*, Vol. SE-5, No.5, September 1979, 440-452.

[13] Chen, C.H., Ho, Y.C., An Approximation Approach of the Standard Clock Method for General Discrete Event Simulation, *IEEE Transactions on Control Systems Technology*, Vol. 3, No. 3, 1995, 309-317.

[14] Chen, F. F., Hardware-Based Simulator For Flexible Manufacturing Systems, *NSF Design and Manufacturing Grantees Conference*, Jan., 1998, Monterrey, Mexico, 125-126.

[15] Chen, F. F., Xu, D., Microprocessor-based Simulator for Flexible Manufacturing Systems, *Proceedings of 1999 NSF Design and Manufacturing Grantees Conference*, CA, January 1999.

[16] Chen, F. F., Xu, D., Enabling Hardware Technologies For Real-time Simulation And Control Of Flexible Manufacturing Systems, *Proceedings of the 9th International FAIM Conference,* Tilburg, Netherlands, June 1999, 737-748.

[17] Chen, M., Mathematical Programming Model For AGVS Planning and Control In Manufacturing Systems, *Computers & Industrial Engineering*, Vol. 30, No. 4, Sept. 1996, 647-658.

[18] Cheng, X. G., Kimbler, D. L., A Stochastic Scheduling Model for Flexible Manufacturing Systems, *Japan/USA Symposium on Flexible Automation*, Vol. 2, 1996, 1301-1304.

[19] Cheok, K. C., Huang, N., and et al., Concurrent Real-Time Simulation and Animation of Active Suspension Control Systems Using Digital Signal Processor and Graphics Hardware., *Trasportation Systems,* Vol. 44, 359-366.

[20] Chuter, C., Jernigan, S.R., and Barber K. S., A Virtual Environment Simulator for Reactive Manufacturing Schedules, *Symposium on Virtual Reality in Manufacturing Research and Education*, Oct., 1996, 197-209.

[21] Cortellessa, V. A., Lazeolla, G. A., Performance Analysis of Optimistic Parallel Simulations With Limited Rolled Back Events, *Simulation Practice and Theory,* Vol. 7, No. 4, June 1999, 325-347.

[22] Cosic, K., Kopriva, I., and Sikic, T., A Methodology For Digital Real Time Simulation of Dynamic Systems Using Modern DSPs, *Simulation Practice & Theory,* Vol. 5, No. 2, 1997, 137-151.

[23] Dado, B., Menhart, P., and Safarik, J., Distributed Simulation: A Simulation System for Discrete Event Systems, *Decentralized and Distributed Systems* (A-39), 1993, 343-353.

[24] Davis, W. J., Macro, J. G., Brook, A. L., Lee, M. S., Zhou, G. S., Developing a Real-Time Emulation/Simulation Capability for the Control Architecture to the RAMP FMS, *Proceedings of the 1996 Winter Simulation Conference*, Coronado, WSC'96, CA, USA, Dec.1996, 171-177.

[25] Devanathan, R., Control Loop Simulation of Flexible Manufacturing System, *IEEE International Symposium on Circuit and Systems*, 1991, 2721-2724.

[26] Drake, G. R., Smith, J. S., Simulation System For Real-Time Planning, Scheduling, and Control, *Proceedings of the 1996 Winter Simulation Conference*, Coronado, WSC'96, CA, USA, Dec.1996, 1083-1090.

[27] Drake, G. R., Smith, J. S., Simulation As a Planning and Scheduling Tool for Flexible Manufacturing Systems, *Proceedings of 1995 Winter Simulation Conference*, 1995, 805-811.

[28] Duffie, N. A., Prabhu, V. V., Real-Time Distributed Scheduling of Heterarchical Manufacturing Systems, Journal of Manufacturing Systems, Vol. 13, No. 2, 1994, 94-107.

[29] ElMaraghy, H. A., Abdallah, I. B., and EIMaraghy, W. H., On-Line Simulation and Control in Manufacturing Systems, *Annals of the CIRP: Manufacturing Technology*, Vol. 47, No. 1, 1998, 401-404.

[30] Fanti, M. P., Piscitelli, G., and Turchiano, B., System Approach to Design Generic Software for Real-Time Control of Flexible Manufacturing Systems, *IEEE Transactions on Systems, Man, & Cybernetics Part A: Systems & Humans,* Vol. 26, No. 2, Mar. 1996, 190-202.

[31] Farrington, P. A., Feng, Y., Simulators as a Tool for Rapid Manufacturing Simulation, *Proceedings of the 1994 Winter Simulation Conference*, SWC'94, Dec. 1994, 994-999.

[32] Fishwick, P. A., Simulation Model Design and Execution: Building Digital Worlds, *Prentice Hall,* 1995, ISBN 0130986097.

[33] Fujii, S., Sandoh, H., Matsuda, H., Tasaka, M., A Study on Distributed Simulation for Flexible Manufacturing Systems, *IFAC information Control problem in Manufacturing technology*, Madrid, Spain, 1989.

[34] Fujii, S., Tsunoda, H., Ogita, A., Kidaniand, Y., Distributed Simulation Model for Computer Integrated Manufacturing, *Winter Simulation Conference Proceedings 1994,* 946-953.

[35] Fujii, S., Kidani, Y., Ogita, A., Kaihara, T., Synchronization Mechanisms for Integration of Distributed Manufacturing Simulation Systems, *Simulation,* Vol. 72, No. 3, 1999, 187-197.

[36] Fujimoto, R. M., Parallel and Distributed Discrete Event Simulation: Algorithms and Applications, *Proceedings of the 1993 Winter Simulation Conference*, 1993,106-114.

[37] Fujimoto, R. M., Parallel and Distributed Simulation Systems, John Wiley & Sons, Inc., New York, 2000, ISBN 0-471-18383-0.

[38] Gershwin, S. B., Design and Operation of Manufacturing Systems Control and System Theoretical Models and Issues, *Proceedings of the American Control Conference,* Vol. 3, 1997, IEEE, Piscataway, NJ, USA, 1909-1913.

[39] Gonzalez, F. G., A Simulation-Based Controller Builder for Flexible Manufacturing Systems, *Proceedings of the 1996 Winter Simulation Conference*, Coronado, WSC'96, CA, USA, Dec.1996, 1068-1075.

[40] Gonzalez, F. G., Davis, W. J., Simulation-Based Controller for Distributed Discrete-Event Systems With Application to Flexible Manufacturing, *Winter Simulation Conference Proceedings 1997,* Piscataway, NJ, USA, 1997, 845-852.

[41] Gonzalez, L. R., Garcia, M. L., and Centeno, M. A., on-Line Knowledge-Based Simulation for FMS: A State of the Art Survey, *Proceedings of the 1996 Winter Simulation Conference*, Coronado, WSC'96, CA, USA, Dec. 1996, 1057-1061.

[42] Hamilton, J. A. Jr., Nash, D. A., Pooch, U. W., Distributed Simulation, *CRC Press*, Boca Raton, New York, 1997, ISBN 0-8493-2590-0.

[43] Harmonosky, Catherine M., and Robohn, S. F., Investigating the Application Potential of Simulation to Real-Time Control Decisions, *International Journal of Computer Integrated Manufacturing*, Vol. 8, No.2, 1995, 126-132.

[44] Harmonosky, G.M., Implementation Issues Using Simulation for Real-Time Scheduling, Control, and Monitoring, *Proceedings of 1990 Winter Simulation Conference*, 1990, 595-598.

[45] Harrell, C. R., ProModel Tutorial, *Proceedings of the 1990 Winter Simulation Conference*, 1990, 128-131.

[46] Heidelberger, P., Statistical Analysis of Parallel Simulations, Proceedings of 1986 Winter Simulation Conference, Dec. 1986, 290-295.

[47] Ho, Y.C., Cassandras, C. G. , and Makhlouf, M., Parallel Simulation of Real Time Systems Via Standard Clock Approach, *Mathematics and Computers in Simulation*, Vol. 35, 1993, 33-41.

[48] Hon, K. K. B., Ismail, H.S., Application of Transputers for Simulation of Manufacturing Systems-A Preliminary Study, *Journal of Engineering Manufacture*, Vol.205, No.1, 1991, 19-23.

[49] Huang, H. P., Chang, P. C., Sepcification, Modelling and Control of a Flexible Manufacturing Cell, *International Journal of Production Research*, Vol.30, No.11, 1992, 2515-2543.

[50] Huang, N., Cheok, K. C., Horner, T. G., Settle, T., Real-Time Simulation and Animation of Suspension Control System Using TI TMS320C30 Digital Signal Processor, *Simulation*, Vol. 61, No. 6, Dec. 1993, 405-416.

[51] Jefferson, D. R., Virtual Time, *ACM Transactions on Programming Language and Systems*, Vol. 7, No. 3, July 1985, 404-425.

[52] Jeong, K. C., Kim, Y. D., Real-Time Scheduling Mechanism for A Flexible Manufacturing System: Using Simulation and Dispatching Rules, *International Journal of Production Research*, Vol. 36, No. 9, Sept. 1998, 2609-2626.

[53] Kamath, M. P., Mize, D. B., and Joe, H., A Comprehensive Modeling and Analysis Environment for Manufacturing Systems, *Industrial Engineering Research - Conference Proceedings 1995,* IIE, Norcross, GA, USA, 759-768.

[54] Kim, M. H., Kim, Y. D., Simulation-Based Real-Time Scheduling in a Flexible Manufacturing System, *Journal of Manufacturing Systems*, Vol. 13, No. 2, 1994, 85-93.

[55] Konas, P., Yew, P. G., Parallel Simulations of Multiprocessors, *Simulation Practice & Theory,* Vol. 1, No. 6, July 1994, 249-266.

[56] Kosturiak, J., Gregor, M., FMS Simulation: Some Experience and Recommendations, *Simulation Practice and Theory*, Vol. 6, Issue 5, July 15, 1998, 423-442.

[57] Krasnikov, V. F., Simulation of Automated Manufacturing Systems, *Mechanism and Machine Theory,* Vol. 31, No. 5. 1996, 691-704.

[58] Kunnathur, A. S., Sampath, S., Dynamic Rescheduling of A Job Shop: A Simulation Study, *Proceedings of the 1996 Winter Simulation Conference*, Coronado, WSC'96, CA, USA, Dec. 1996, 1091-1097.

[59] Kurugollu, F., Palaz, H., and et al., Advanced Educational Parallel DSP System Based on TM320C25 Processors, *Microprocessors and Microsystems*, Vol. 19, No.3, April 1995, 147-155.

[60] Lavoie, M., Que-Do, V., and et al., Real-Time Simulation of Power System Stability Using Parallel Digital Signal Processors, *Mathematics & Computers in Simulation,* Vol. 38, No. 4-6 Aug. 1995, 283-292.

[61] Law, A. M., Models of Random Machine Downtimes for Simulation, *Proceedings of the 1990 Winter Simulation Conference*, 1990, 314-316.

[62] Law, A. M., Kelton, W. D., Simulation Modeling and Analysis, *McGraw-Hill*, *Inc.,* New York, 1999, 3rd, ISBN 0-07-059292-6.

[63] L'Ecuyer, P., Andres, T.H., A Random Number Generator Based on the Combination of Four LCGs, *Mathematics and Computers at Simulation*, Vol. 44, Issue 1, May 1997, 99-107.

[64] Lee, D. Y., Dicesare, F., Scheduling of Flexible Manufacturing Systems Employing Automated Guided Vehicles, *Advances in Manufacturing Systems*, 1994, 139-144.

[65] Lee, Y. H., Iwata, K., Part Ordering Through Simulation-Optimization at An FMS, *International Journal of Production Research,* Vol. 29, No. 7, 1991, 1309-1323.

[66] Lim, C. C., Low, Y. H., and et al., Implementations of Dispatching Rules in Parallel Manufacturing Simulation, *Proceedings of the 1998 Winter Simulation Conference*, Dec. 1998, 1591-1596.

[67] Lynggaard, H. J., Bilberg, A., and Alting, L., New Concepts and Methods for Developing Shop Floor Control Systems, *Annals of the CIRP: Manufacturing Technology*, Vol. 47, No. 1, 1998, 377-380.

[68] Mascarenhas, E., Knop, F., and Rego, V., Parasol: A Multithreaded System for Parallel Simulation Based on Mobile Threads, *Proceedings of the 1995 Winter Simulation Conference*, 1995, 690-697.

[69] McConnell, P. G., Medeiros, D. J., Real-Time Simulation for Decision Support at Continuous Flow Manufacturing Systems, *Proceedings of the 1992 Winter Simulation Conference*, 1992, 936-944.

[70] Merchawi, N. S., ElMaraghy, H. A., Analytical Approach for Using Simulation at Real-Time Decision Making at FMSs, *Journal of Manufacturing Systems,* Vol. 17, No. 6, 1998, 418-435.

[71] Morito, S., Lee, K. H., Development of A Simulation-Based Planning System for A Flexible Manufacturing System, *Proceedings of the 1992 Winter Simulation Conference*, 1992, 908-915.

[72] Muller, D. J., Jackman, J. K., and Fitzwater, C., A Simulation-Based Work Order Release Mechanism for A Flexible Manufacturing System, *Proceedings of the 1990 Winter Simulation Conference*, 1990, 599-602.

[73] Niederreiter, H., Some Linear and Nonlinear Methods for Pseudorandom Number Generation, *Proceedings of the 1995 Winter Simulation Conference*, 1995, 250-254.

[74] Oertel, C.H., Gelhaar, G., A Fast Real-Time Simulation of A Complex Mechanical System on A Parallel Hardware Architecture, *International Telemetering Conference (Proceedings),* Vol.28, 1992, 441-448.

[75] Olivera, H., Scheduling at Job Shops With Machine Breakdowns: An Experimental Study, *Computers & Industrial Engineering,* Vol. 36, Issue 1, Jan. 1999, 137-162.

[76] Page, E. H., Nance, R. E., Parallel Discrete Event Simulation: A Modeling Methodological Perspective, *Proceedings of 8th workshop parallel distributed simulation*, 1994, 88-93.

[77] Paker, Y., Multi-Microprocessor Systems, *Academic Press Inc.*, 1983, Lodon, New York, ISBN 0-12-543980-6.

[78] Pargas, R. P., Peck, J. C. and Khambekar, P. K., Near-Term Distributed Simulation of Apparel Manufacturing, Proceedings of the 1990 Winter Simulation Conference, 1990, 614-618.

[79] Park, J., Park, J. W., Woo, J., Kim, J., Kwonand, W. H., An Object-Based Run-Time Executive for Control of Flexible Manufacturing Systems, *Manufacturing Research and Technology,* Vol. 22, 1995, 103-108.

[80] Peng, C., Chen, F.F., Parallel Discrete Event Simulation of Manufacturing Systems: A Technology Survey, *Computers and Industrial Engineering*, Vol. 31, No. 2, 1996, 327-330.

[81] Peters, B. A., Smith, J. S., and et al., Advanced Tutorial - Simulation-Based Scheduling and Control, *Proceedings of the 1996 Winter Simulation Conference*, Coronado, WSC'96, CA, USA, Dec.1996, 194-198.

[82] Phillips, C. L., Cuthbert, L. G., Concurrent Discrete Event-Driven Simulation Tools, *IEEE Journal on Selected Areas in Communications*, Vol. 9, No.3, April 1991, 477-485.

[83] Platzner, M., Steger, C. H., and Weiss, R., Experimental Evaluation of Multi-DSP Architectures at High Performance Applications, Mediterranean Electrotechnical Conference-MELECON 3, 1994, 1123-1126.

[84] Quinnell, R. A., The Ever-Evolving Vmebus Adapts to User Needs, *EDN*, Feb. 1997, 82-95.

[85] Rahman, M. F., Microprocessor and Programmable Controller Based Industrial Automation, *Computer-Based Automation*, 1985, 507-518.

[86] Ramzi, B., Mohamed, B., and Georges, H., An Object Model for Simulation of Manufacturing Systems, *Manufacturing Research and Technology,* Vol. 22: Advances in Manufacturing Systems: Design, Modeling and Analysis, 1995, 137-142.

[87] Righter, R., Walrand, R. C., Distributed Simulation of Discrete Event Systems, *Proceedings of the IEEE*, Vol.77, No.1, January 1989, 99-113.

[88] Roberts, E.A., Shires, N., The Application of Multiprocessing to Simulation, *Proceedings of the 1st International Conference on Simulation in Manufacturing,* Kempston, English, 1985, 85-96.

[89] Rogers, P., Gordon, R. J., Simulation for Real-Time Decision Making at Manufacutring Systems, *Proceedings of the 1993 Winter Simulation Conference*, 1993, 866-874.

[90] Ruppel, F., Wysor, W., Mighty Microprocessors Boost Process Simulation, *InTech*, Sept. 1997, 69-72.

[91] Sammons, S. M., Cochran, J. K., The Use of Simulation at the Optimization of A Cellular Manufacturing System, *Proceedings of the 1996 Winter Simulation Conference*, Dec.1996, 1129-1133.

[92] Schöf, S., Efficient Data Structures for Time Warp Simulation Queues, *Journal of Systems Architecture,* Vol. 44, Issue: 6-7, March 1998, 497-517.

[93] Schriber, T. J., Brunner, D. T., Inside Discrete-Event Simulation Software: How It Works and Why It Matters, *Winter Simulation Conference Proceedings,* Vol. 1, 1998, 77-85.

[94] Shukla, C. H., Chen, F. F., The State of the Art at Intelligent Real-Time FMS Control: A Comprehensive Survey, *Journal of Intelligent Manufacturing*, Vol. 7, 1996, 441-455.

[95] Sieveking, A., Davis, W. J., An Object-Oriented Simulation Language for Master Production Scheduling at A Flexible Manufacturing Environment, *Manufacturing Research and Technology,* Vol. 22: Advances in Manufacturing Systems: Design, Modeling and Analysis, 1995, 189-194.

[96] Smith, G. D., Medeiros, D. J., Simulation of Flexible Control Strategies, *Proceedings of the 1995 Winter Simulation Conference*, Dec. 1995, 799-803.

[97] Spasov, P., Microcontroller Technology: the 68HC11, 3th ed., *Prentice Hall Inc.*, 1996, ISBN 0-13-901240-0.

[98] Stankovic, J. A., Ramamritham, K., and Zlokapa, G., Real-Time Platforms and Environments for Time Constrained Flexible Manufacturing, *IEEE,* 1994, 96-99.

[99] Stanley, M. C., Colgate, J. E., Computer Simulation of Interacting Dynamic Mechanical Systems Using Distributed Memory Parallel Processors, Advances in Robotics, Vol.42, 1992, p55-61

[100] Susumu, F. J., Hirano, M., A Basic Study on Distributed Simulation Model of Virtual Manufacturing System Under CIM Environment, *Production Research,* 1993, 281-282.

[101] TMS320C3x DSP Starter Kit User's Guide, Texas Instrument Incorporated, 1996.

[102] Thomson, N., Simulation at Manufacturing, John *Wiley& Sons Inc.,* 1995, ISBN 0-471-957380.

[103] Throckmorton, P. J., Wozniak, L., A Generic DSP-Based Real-Time Simulator With Application to Hydrogenerator Speed Control Development, *IEEE Transactions on Energy Conversion*, Vol. 9, No. 2, Jun. 1994, 238-242.

[104] Timpelmeier, H., Kuhn, H., Flexible Manufacturing Systems: Decision Support for Design and Operation, *John Wiley& Sons Inc.*, New York, 1993, ISBN 0-471-30721-1.

[105] Tirpak, T. M., Daniel, S. M. and et al., A Note on A Fractal Architecture for Modeling and Controlling Flexible Manufacturing Systems, *IEEE Transactions on Systems, Man & Cybernetics,* Vol. 22, No. 3, May-Jun 1992, 564-567.

[106] Tokhi, M. O., Hossian, M. A., and Chambers, C., Performance Evaluation of DSP and Transputer Based Systems at Sequential Real-Time Applications, *Microprocessors and Microsystems*, Vol. 21, 1997, 237-248.

[107] Vakili, P., Massively Parallel and Distributed Simulation of A Class of Discrete Event Systems: A Different Perspective, *ACM Transactions on Modeling and Computer Simulation*, Vol. 2, No. 3, 1992, 214-238.

[108] Wang, J. J., Abrams, M., Approximate Time-Parallel Simulation of Queuing Systems with Losses, Proceedings of the 1992 Winter Simulation Conference, Dec. 1992, 700-708.

[109] Wang, J. J., Efficient Parallel Simulations and Their Application to Communication Networks, Dissertation of the Virginia Polytechinic Institute and State University, 1994.

[110] Watson, E. F., Sadowski, R. P., Developing and Analyzing Flexible Cell Systems Using Simulation, *Proceedings of the 1994 Winter Simulation Conference*, Dec. 1994, 978-985.

[111] Wells, B. E., Ricks, K. G. and Weir, J. M., Parallel Simulation of A Large-Scale Aerospace System at A Multicomputer Environment, *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 33, No. 2, April 1997, 507-521.

[112] Will, P., Fossum, B. M., and et al., Information Technology for Manufacturing: A Research Agenda, *National Academy Press,* Washington, D. C., 1995, ISBN 0-309-05179-7.

[113] Wysk, R. A., Co, H. C., Robusteness of CAN-Q in Modelling Aotomated Manufacturing Systems, *International Journal of Production Research*, Vol. 24, No. 6, Nov-Dec, 1986, 1485-1503.

[114] Xu, D., Chen, F. F., An Integrated Decomposition Method for Real-time Rescheduling of Flexible Manufacturing Systems, *Proceedings of 1999 International Conference on Advanced manufacturing Technology*, Xi'an, China, June 1999, 450-454.

[115] Zhou, M., Venkatesh, K., Modeling, Simulation, and Control of Flexible Manufacturing Systems: A Petri Net Approach, *World Scientific Pub. Co.,* 1999, ISBN 981023029X.

APPENDIX A: SOFTWARE-BASED SIMULATION RESULTS

A.1 Simulation Results for the Parallel Production Lines

```
-------------------------------------------------------------------------------
General Report
Output from E:\Users\dxu\Filestore\FMSModel\ThesisModel\Production_Lines2.MOD
Date: Feb/19/2000   Time: 09:55:31 PM
-------------------------------------------------------------------------------
Scenario        : Normal Run
Replication     : Average
Period          : Final Report (16 hr to 256 hr Elapsed: 240 hr)
Warmup Time     : 16 hr (Std. Dev. 0 sec)
Simulation Time : 256 hr
-------------------------------------------------------------------------------

LOCATIONS
                                        Average
Location  Scheduled           Total     Minutes    Average   Maximum   Current
Name        Hours   Capacity  Entries  Per Entry  Contents  Contents  Contents  % Util
--------- ---------  -------- -------  ---------  --------- --------  --------  ------
Lathe1         240     1        0   0.000000            0         0         0    0.00  (Average)
Lathe1           0     0        0   0.000000            0         0         0    0.00  (Std. Dev.)
Mill1          240     1   1336.8   4.366875     0.405391         1       0.4   40.54  (Average)
Mill1            0     0  18.8432   0.013784   0.00583826         0  0.516398    0.58  (Std. Dev.)
Mill2          240     1   1141.1   8.892592     0.704673         1       0.6   70.47  (Average)
Mill2            0     0  7.43042   0.015551   0.00413765         0  0.516398    0.41  (Std. Dev.)
Lathe2         240     1   1141.2   9.059973     0.718004         1       0.9   71.80  (Average)
Lathe2           0     0  7.45058   0.012785   0.00502526         0  0.316228    0.50  (Std. Dev.)

RESOURCES

                                Average   Average   Average
                         Number Minutes   Minutes   Minutes
Resource         Scheduled Of Times Per       Travel    Travel  % Blocked
Name        Units  Hours    Used     Usage     To Use    To Park  In Travel  % Util
------------   ----------  -------- --------  --------  --------  ---------  ------
Gryphon1        1   240    2672.6    1.544996  0.264963  0.000000       0.00   33.59  (Average)
Gryphon1        0     0   37.7395    0.000016  0.000067  0.000000       0.00    0.47  (Std. Dev.)
Gryphon2        1   240    3422.1    1.829625  0.189993  0.000000       0.00   47.99  (Average)
Gryphon2        0     0   22.8008    0.000240  0.000041  0.000000       0.00    0.32  (Std. Dev.)


ENTITY ACTIVITY

                          Average    Average    Average    Average   Average
                  Current    Minutes    Minutes    Minutes    Minutes   Minutes
Entity    Total   Quantity         In    In Move   Wait For         In
Name      Exits  In System     System      Logic  Res, etc.  Operation   Blocked
------  -------  ---------  ---------  ---------  ---------  ---------  --------
PartA      1199        3.7  41.208983  16.062383   0.075371  16.425494  8.645734  (Average)
PartA   10.9646   0.483046   0.458920   0.194795   0.037935   0.141191  0.126512  (Std. Dev.)
PartB    1397.2        1.1  11.861616   4.133209   0.290919   7.032244  0.405243  (Average)
PartB   17.7564   0.567646   0.413116   0.147240   0.113478   0.116734  0.068921  (Std. Dev.)

ENTITY STATES BY PERCENTAGE
             %          %
Entity  In Move   Wait For             %        %
Name      Logic  Res, etc.  In Operation  Blocked
------  -------  ---------  ------------  -------
PartA     38.98       0.18         39.86    20.98  (Average)
PartA      0.13       0.09          0.16     0.11  (Std. Dev.)
PartB     34.84       2.43         59.32     3.41  (Average)
PartB      0.10       0.88          1.11     0.51  (Std. Dev.)
```

VARIABLES

Variable Name	Total Changes	Average Minutes Per Change	Minimum Value	Maximum Value	Current Value	Average Value	
Total Parts Num A	1199.3	12.002539	82.1	1281.4	1281.4	682.635	(Average)
Total Parts Num A	11.373	0.111895	1.66333	12.2038	12.2038	8.86208	(Std. Dev.)
Total Parts Num B	1397.4	10.303998	94.2	1491.6	1491.6	791.659	(Average)
Total Parts Num B	18.0012	0.130108	5.55378	19.5061	19.5061	11.155	(Std. Dev.)

A.2 Simulation Results for the Manufacturing Cell

```
-------------------------------------------------------------------------------
General Report
Output from E:\Users\dxu\Filestore\FMSModel\ThesisModel\Fms_thesis.MOD [DesktopFMS]
Date: Feb/19/2000   Time: 08:53:21 PM
-------------------------------------------------------------------------------
Scenario        : Normal Run
Replication     : Average
Period          : Final Report (16 hr to 256 hr Elapsed: 240 hr)
Warmup Time     : 16 hr (Std. Dev. 0 sec)
Simulation Time : 256 hr
-------------------------------------------------------------------------------
```

LOCATIONS

Location Name	Scheduled Hours	Capacity	Total Entries	Average Minutes Per Entry	Average Contents	Maximum Contents	Current Contents	% Util	
Lathe1	240	1	777.9	12.668836	0.684258	1	0.7	68.43	(Average)
Lathe1	0	0	11.1699	0.306487	0.013412	0	0.483046	1.34	(Std. Dev.)
Mill1	240	1	1302.1	6.870607	0.621287	1	0.9	62.13	(Average)
Mill1	0	0	17.0714	0.025158	0.0100793	0	0.316228	1.01	(Std. Dev.)
Mill2	240	1	1357.9	6.930063	0.653488	1	0.4	65.35	(Average)
Mill2	0	0	5.64604	0.029178	0.00219712	0	0.516398	0.22	(Std. Dev.)
Lathe2	240	1	771.8	12.627808	0.676801	1	0.6	67.68	(Average)
Lathe2	0	0	5.86515	0.143115	0.00801149	0	0.516398	0.80	(Std. Dev.)

RESOURCES

Resource Name	Units	Scheduled Hours	Number Of Times Used	Average Minutes Per Usage	Average Minutes Travel To Use	Average Minutes Travel To Park	% Blocked In Travel	% Util	
Gryphon1	1	240	3430.3	1.503832	0.360953	0.000000	0.00	44.42	(Average)
Gryphon1	0	0	28.1229	0.001900	0.001882	0.000000	0.00	0.36	(Std. Dev.)
Gryphon2	1	240	3499.8	1.744622	0.218239	0.000000	0.00	47.70	(Average)
Gryphon2	0	0	10.1412	0.001985	0.000795	0.000000	0.00	0.16	(Std. Dev.)

ENTITY ACTIVITY

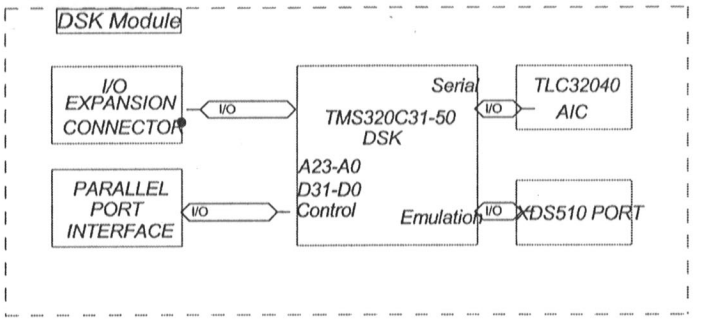
Entity Name	Total Exits	Current Quantity In System	Average Minutes In System	Average Minutes In Move Logic	Average Minutes Wait For Res, etc.	Average Minutes In Operation	Average Minutes Blocked	
PartA	1590.6	4.5	38.575616	12.148464	0.000000	16.689569	9.737583	(Average)
PartA	13.4098	0.707107	0.870145	0.193329	0.000000	0.138621	0.636857	(Std. Dev.)
PartB	1193	1.7	20.896560	8.052692	0.000000	6.860035	5.983833	(Average)
PartB	14.4222	0.948683	0.385084	0.201240	0.000000	0.109863	0.183312	(Std. Dev.)

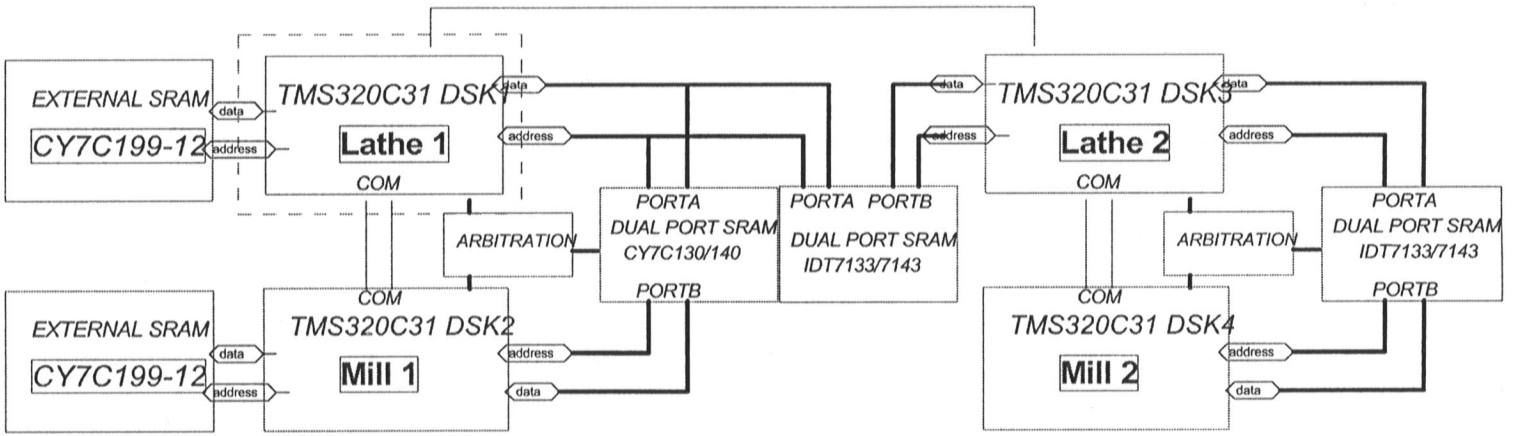
ENTITY STATES BY PERCENTAGE

Entity Name	% In Move Logic	% Wait For Res, etc.	% In Operation	% Blocked	
PartA	31.50	0.00	43.28	25.22	(Average)
PartA	0.39	0.00	0.77	1.12	(Std. Dev.)
PartB	38.53	0.00	32.83	28.63	(Average)
PartB	0.45	0.00	0.53	0.59	(Std. Dev.)

VARIABLES

Variable Name	Total Changes	Average Minutes Per Change	Minimum Value	Maximum Value	Current Value	Average Value	
Total Parts Num A	1591.3	9.047038	107	1698.3	1698.3	905.027	(Average)
Total Parts Num A	12.9104	0.073357	5.71548	13.367	13.367	9.37736	(Std. Dev.)
Total Parts Num B	1192.3	12.070909	81.8	1274.1	1274.1	677.913	(Average)
Total Parts Num B	14.7351	0.146730	2.34758	15.7229	15.7229	6.91794	(Std. Dev.)



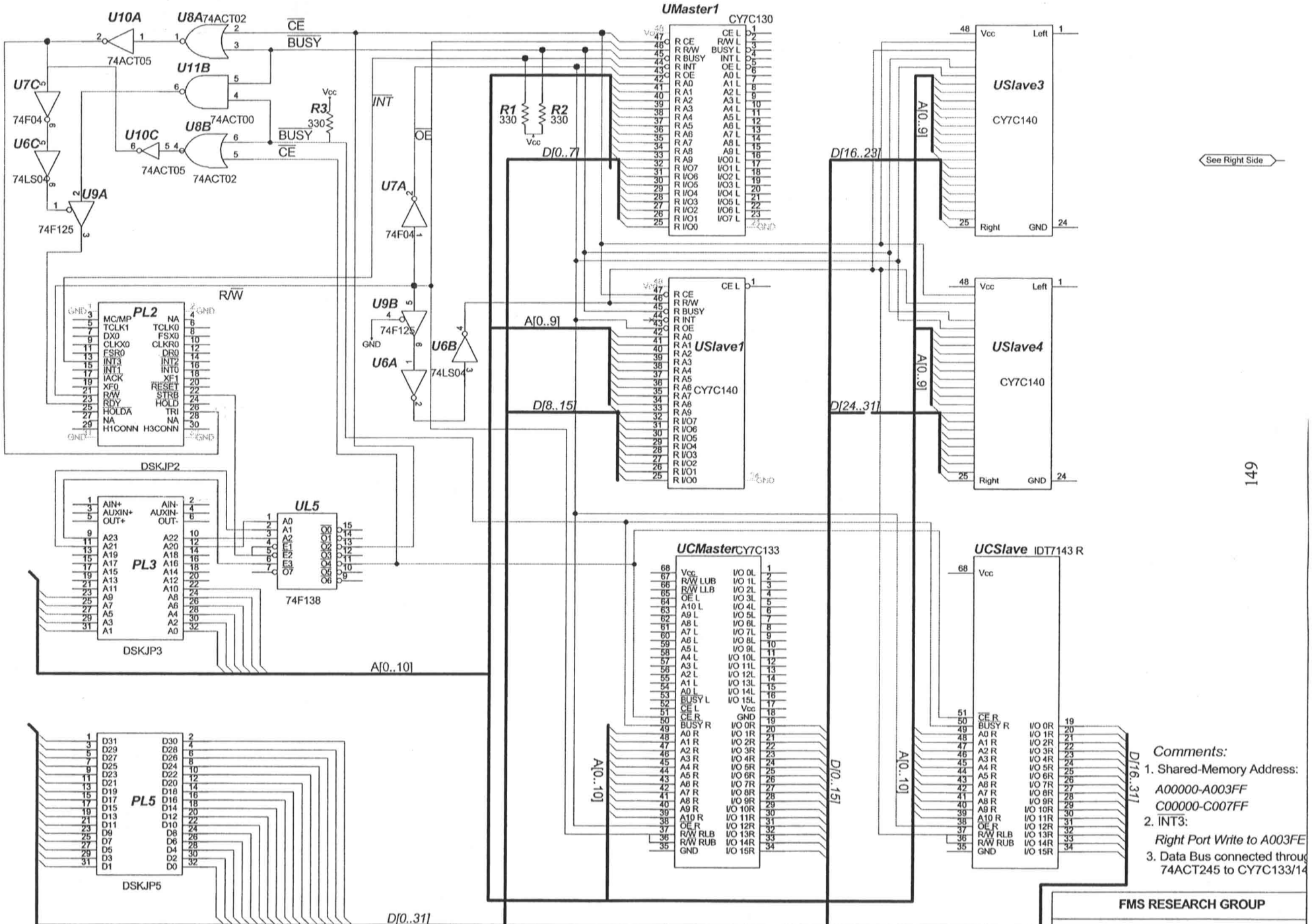


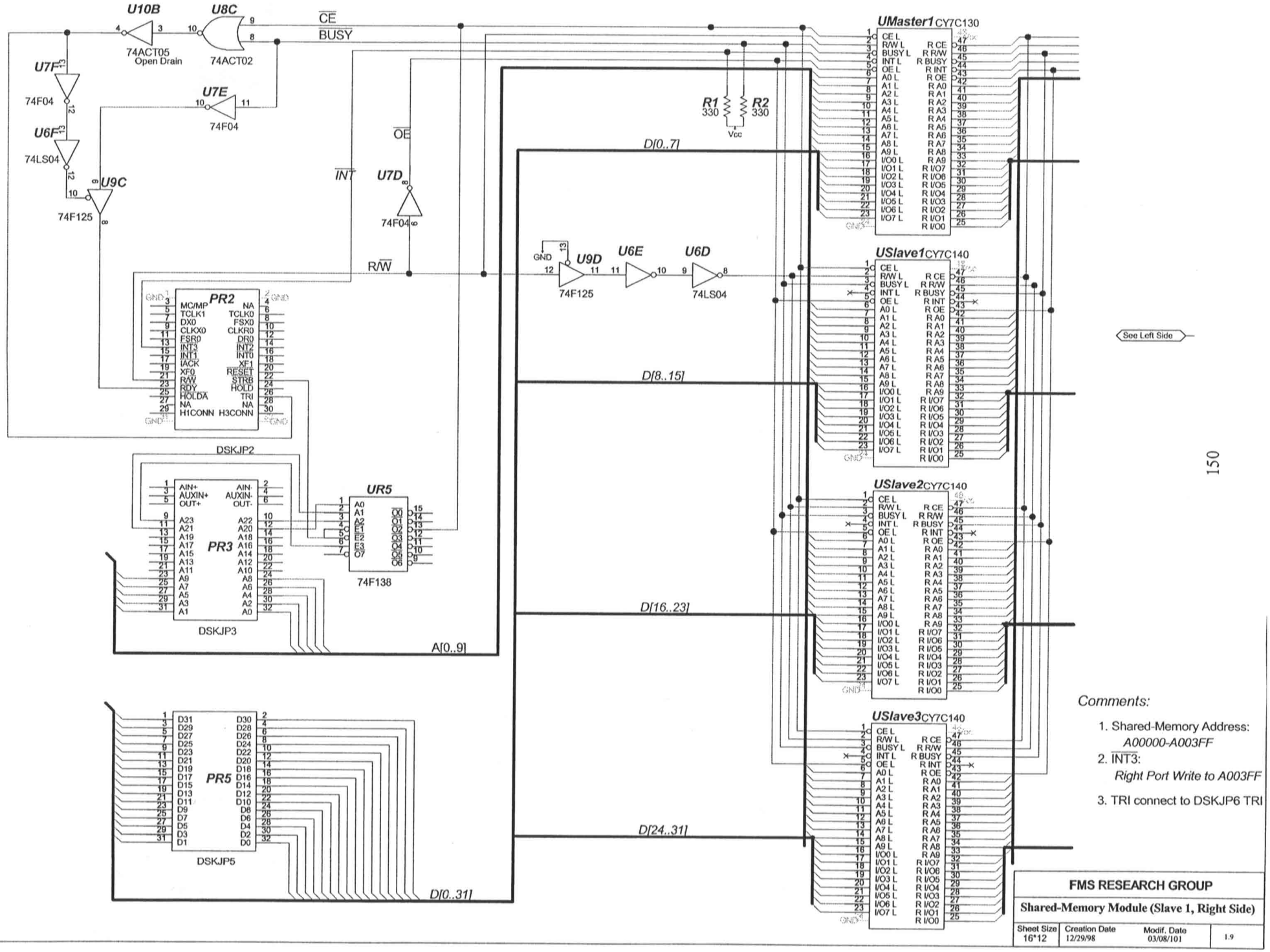
Notes:

1. For schematic of DSK, please see TMS320C3X DSP Starter Kit User's Guide [101]

2. Total 5 sheets for master 1 and 2, slave 1 and 2, and external SRAM

FMS RESEARCH GROUP			
SIMPLIFED ARCHITECTURE			
Sheet Size	Creation Date 07/16/98	Modif. Date 05/23/101	1.5

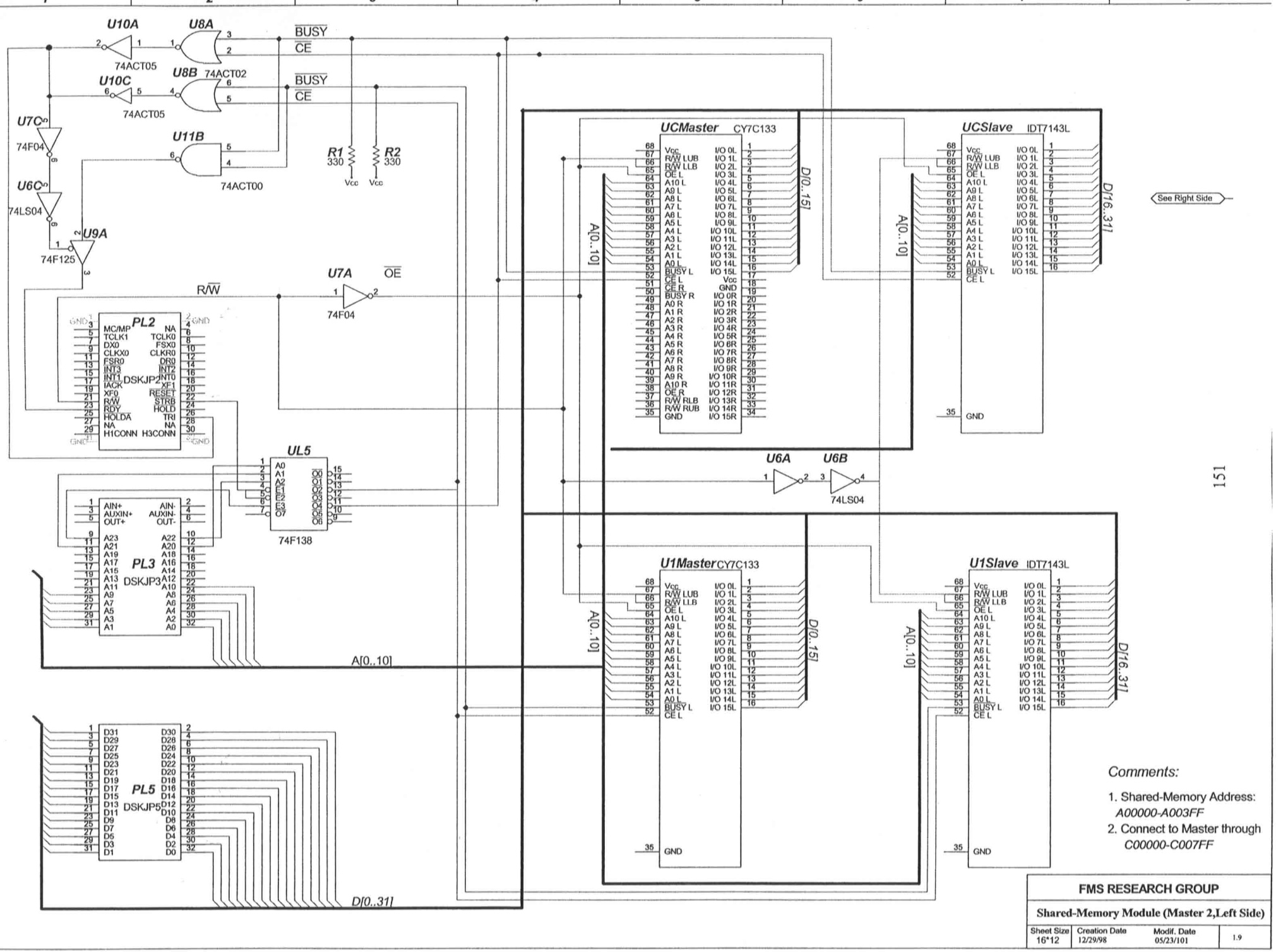

U10A
U8A 74ACT02
74ACT05
CE
BUSY
U11B
U7C
74F04
U6C
74LS04
U10C
74ACT05
U8B 74ACT00
74ACT02
R3
330
Vcc
BUSY
CE
INT
OE
U9A
74F125
R/W
U7A
74F04
R1
330
R2
330
Vcc
D[0..7]
UMaster1
CY7C130
R CE
R R/W
R BUSY
R INT
R OE
R A0
R A1
R A2
R A3
R A4
R A5
R A6
R A7
R A8
R A9
R I/O7
R I/O6
R I/O5
R I/O4
R I/O3
R I/O2
R I/O1
R I/O0
CE L
R/W L
BUSY L
INT L
OE L
A0 L
A1 L
A2 L
A3 L
A4 L
A5 L
A6 L
A7 L
A8 L
A9 L
I/O0 L
I/O1 L
I/O2 L
I/O3 L
I/O4 L
I/O5 L
I/O6 L
I/O7 L
GND
D[16..23]
USlave3
CY7C140
Vcc
Left
Right
GND
A[0..9]
See Right Side
PL2
MC/MP
TCLK1
DX0
CLKX0
FSR0
INT3
INT1
IACK
XF0
R/W
RDY
HOLDA
NA
H1CONN
TCLK0
FSX0
CLKR0
DR0
INT2
INT0
XF1
RESET
STRB
HOLD
TRI
NA
H3CONN
U9B
74F125
U6B
U6A
74LS04
A[0..9]
D[8..15]
USlave1
CY7C140
CE L
R CE
R R/W
R BUSY
R INT
R OE
R A0
R A1
R A2
R A3
R A4
R A5
R A6
R A7
R A8
R A9
R I/O7
R I/O6
R I/O5
R I/O4
R I/O3
R I/O2
R I/O1
R I/O0
GND
USlave4
CY7C140
D[24..31]
DSKJP2
PL3
AIN+
AUXIN+
OUT+
AIN-
AUXIN-
OUT-
A23
A21
A19
A17
A15
A13
A11
A9
A7
A5
A3
A1
A22
A20
A18
A16
A14
A12
A10
A8
A6
A4
A2
A0
DSKJP3
UL5
74F138
A0
A1
A2
G1
G2
G3
O7
O0
O1
O2
O3
O4
O5
O6
A[0..10]
UCMaster CY7C133
Vcc
R/W LUB
R/W LLB
OE L
A10 L
A9 L
A8 L
A7 L
A6 L
A5 L
A4 L
A3 L
A2 L
A1 L
A0 L
BUSY L
CE L
CE R
BUSY R
A0 R
A1 R
A2 R
A3 R
A4 R
A5 R
A6 R
A7 R
A8 R
A9 R
A10 R
OE R
R/W RLB
R/W RUB
GND
I/O 0L
I/O 1L
I/O 2L
I/O 3L
I/O 4L
I/O 5L
I/O 6L
I/O 7L
I/O 8L
I/O 9L
I/O 10L
I/O 11L
I/O 12L
I/O 13L
I/O 14L
I/O 15L
Vcc
GND
I/O 0R
I/O 1R
I/O 2R
I/O 3R
I/O 4R
I/O 5R
I/O 6R
I/O 7R
I/O 8R
I/O 9R
I/O 10R
I/O 11R
I/O 12R
I/O 13R
I/O 14R
I/O 15R
A[0..10]
D[0..15]
UCSlave IDT7143 R
Vcc
CE R
BUSY R
A0 R
A1 R
A2 R
A3 R
A4 R
A5 R
A6 R
A7 R
A8 R
A9 R
A10 R
OE R
R/W RLB
R/W RUB
GND
I/O 0R
I/O 1R
I/O 2R
I/O 3R
I/O 4R
I/O 5R
I/O 6R
I/O 7R
I/O 8R
I/O 9R
I/O 10R
I/O 11R
I/O 12R
I/O 13R
I/O 14R
I/O 15R
A[0..10]
D[16..31]
PL5
D31
D29
D27
D25
D23
D21
D19
D17
D15
D13
D11
D9
D7
D5
D3
D1
D30
D28
D26
D24
D22
D20
D18
D16
D14
D12
D10
D8
D6
D4
D2
D0
DSKJP5
D[0..31]
Comments:
1. Shared-Memory Address:
A00000-A003FF
C00000-C007FF
2. INT3:
Right Port Write to A003FE
3. Data Bus connected throug
74ACT245 to CY7C133/14
FMS RESEARCH GROUP
Shared-Memory Module (Master 1, Left Side)
Sheet Size 16*12
Creation Date 12/29/98
Modif. Date 05/23/101
1/6
149

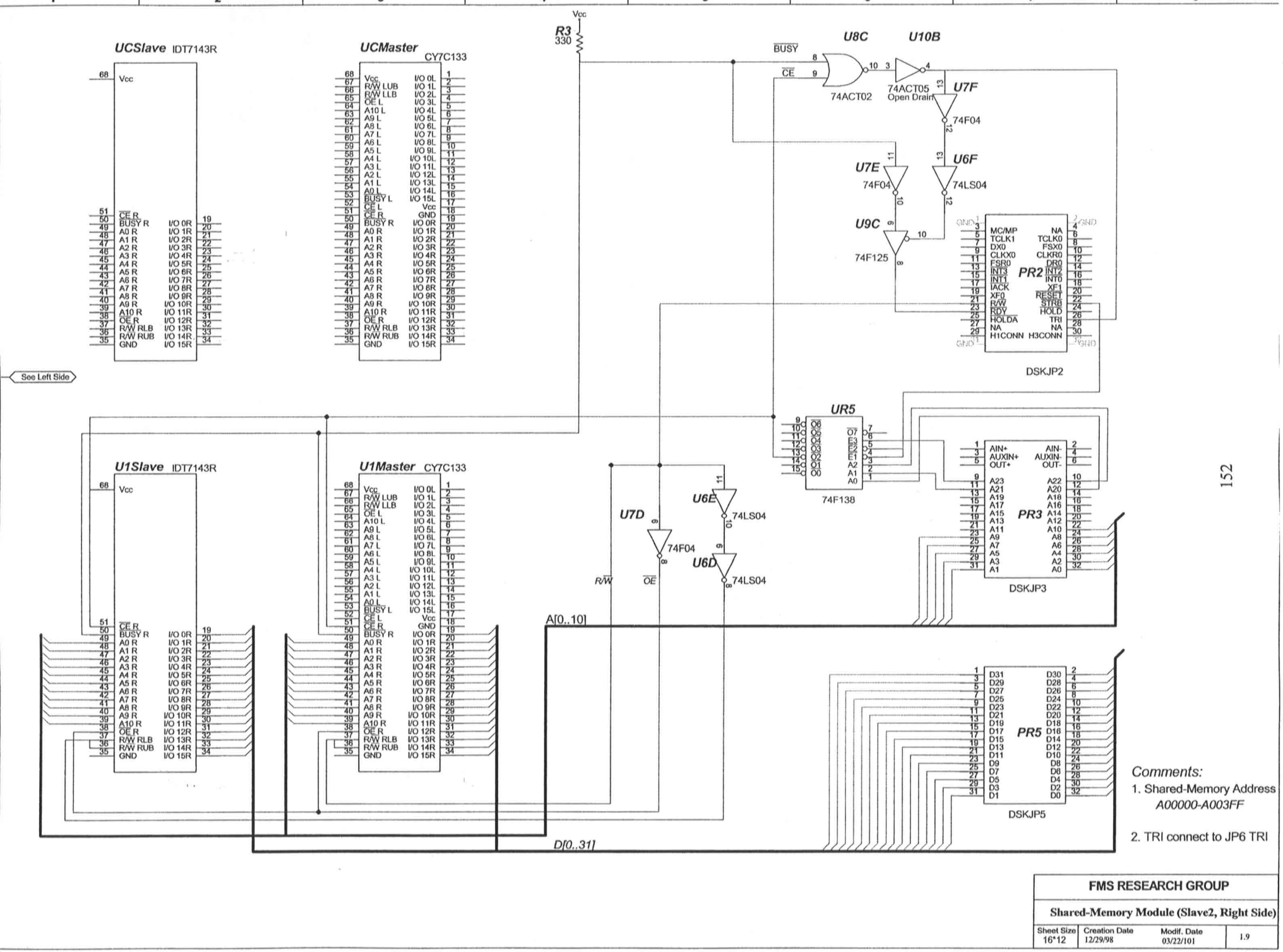


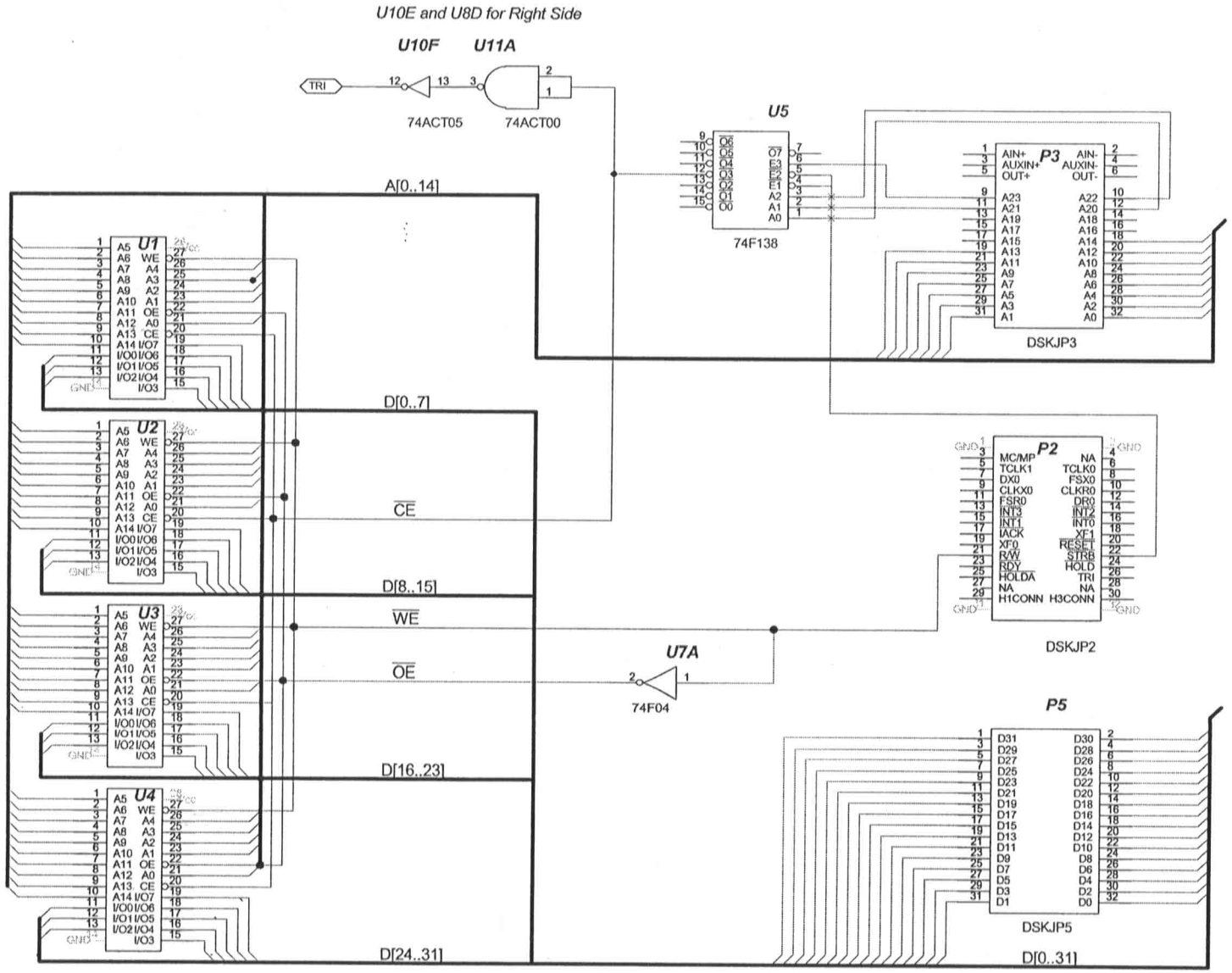
Comments:

1. Shared-Memory Address:
   A00000-A003FF
2. INT3:
   Right Port Write to A003FF
3. TRI connect to DSKJP6 TRI

FMS RESEARCH GROUP			
Shared-Memory Module (Slave 1, Right Side)			
Sheet Size 16*12	Creation Date 12/29/98	Modif. Date 03/08/101	1.9


U10A
74ACT05
U8A
U8B 74ACT02
U10C
74ACT05
U7C
74F04
U6C
74LS04
U9A
74F125
U11B
74ACT00
R1 330
R2 330
Vcc Vcc
BUSY
CE
BUSY
CE
R/W
U7A
OE
74F04
PL2
DSKJP2
PL3
DSKJP3
PL5
DSKJP5
UL5
74F138
A[0..10]
D[0..31]
UCMaster CY7C133
UCSlave IDT7143L
U1Master CY7C133
U1Slave IDT7143L
D[0..15]
D[16..31]
A[0..10]
U6A U6B
74LS04
See Right Side
151
Comments:
1. Shared-Memory Address:
A00000-A003FF
2. Connect to Master through
C00000-C007FF
FMS RESEARCH GROUP
Shared-Memory Module (Master 2,Left Side)
Sheet Size 16*12
Creation Date 12/29/98
Modif. Date 05/23/101
1.9


UCSlave IDT7143R
UCMaster CY7C133
R3 330
Vcc
U8C
U10B
74ACT02
74ACT05 Open Drain
U7F
74F04
U7E
74F04
U6F
74LS04
U9C
74F125
PR2
DSKJP2
See Left Side
UR5
74F138
U1Slave IDT7143R
U1Master CY7C133
U7D
74F04
U6E
74LS04
U6D
74LS04
R/W
OE
A[0..10]
PR3
DSKJP3
PR5
DSKJP5
D[0..31]
152
Comments:
1. Shared-Memory Address
A00000-A003FF
2. TRI connect to JP6 TRI
FMS RESEARCH GROUP
Shared-Memory Module (Slave2, Right Side)
Sheet Size 16*12
Creation Date 12/29/98
Modif. Date 03/22/101
1.9



1. External Memory Address:
   B00000-B007FFF
2. *$\overline{CE}$ Must Drive TRI low to Avoid Delay*

FMS RESEARCH GROUP			
EXTERNAL MEMORY MODULE			
Sheet Size 16*12	Creation Date 11/26/98	Modif. Date 03/22/101	1.5

APPENDIX C: DATA AND PROGRAMS

C.1 Experiment Case for 1 Workstation

C.1.1 Simulation Results from ProModel

```
-------------------------------------------------------------------------------
General Report
Output from C:\Project\FMSModel\ThesisModel\SingleMachine-1M1R.MOD [DesktopFMS]
Date: Apr/17/2001   Time: 08:03:13 PM
-------------------------------------------------------------------------------
Scenario        : Normal Run
Replication     : All
Period          : Final Report (0 sec to 240 hr Elapsed: 240 hr)
Simulation Time : 240 hr
-------------------------------------------------------------------------------
LOCATIONS
                                      Average
Location  scheduled              Total   Minutes    Average   Maximum   Current
Name        Hours     Capacity  Entries  Per Entry  Contents  Contents  Contents  % Util
---------  ---------  --------  -------  ---------  ---------  --------  --------  ------
Lathe          240         1     1000   6.083533   0.422468         1         1   42.25  (Rep 1)
Lathe          240         1      973   6.015173   0.406442         1         0   40.64  (Rep 2)
Lathe          240         1      961   5.733906   0.382659         1         0   38.27  (Rep 3)
Lathe          240         1      969   6.004700   0.404066         1         0   40.41  (Rep 4)
Lathe          240         1      955   6.095888   0.404276         1         1   40.43  (Rep 5)
Lathe          240         1      982   5.996648   0.408938         1         0   40.89  (Rep 6)
Lathe          240         1      939   6.108776   0.398343         1         1   39.83  (Rep 7)
Lathe          240         1      994   6.093782    0.42064         1         1   42.06  (Rep 8)
Lathe          240         1      964   6.122641   0.409877         1         1   40.99  (Rep 9)
Lathe          240         1     1004   5.914837   0.412396         1         0   41.24  (Rep 10)
Lathe          240         1      982   5.819079   0.396829         1         0   39.68  (Rep 11)
Lathe          240         1      941   5.766648   0.376834         1         0   37.68  (Rep 12)
Lathe          240         1      925   6.200590   0.398302         1         0   39.83  (Rep 13)
Lathe          240         1      948   5.910015   0.389076         1         0   38.91  (Rep 14)
Lathe          240         1      981   5.890619   0.401298         1         0   40.13  (Rep 15)
Lathe          240         1  967.867   5.983789   0.402163         1  0.333333   40.22  (Average)
Lathe          0           0   23.354   0.139822   0.0126551        0   0.48795   1.27 (Std. Dev.)
Conveyor21     240        20     1000   6.543464   0.454407         6         0    2.27  (Rep 1)
Conveyor21     240        20      973   6.521732    0.44067         7         0    2.20  (Rep 2)
Conveyor21     240        20      961   5.273997   0.351966         7         0    1.76  (Rep 3)
Conveyor21     240        20      969   6.239838   0.419889         8         0    2.10  (Rep 4)
Conveyor21     240        20      955   5.419725   0.359433         9         0    1.80  (Rep 5)
Conveyor21     240        20      982   7.412635   0.505501         7         0    2.53  (Rep 6)
Conveyor21     240        20      940   6.304304   0.411531         8         1    2.06  (Rep 7)
Conveyor21     240        20      994   5.503738   0.379911         6         0    1.90  (Rep 8)
Conveyor21     240        20      965   6.030567   0.404132         5         1    2.02  (Rep 9)
Conveyor21     240        20     1004   6.861449   0.478395         9         0    2.39  (Rep 10)
Conveyor21     240        20      983   6.269767   0.427999         7         0    2.14  (Rep 11)
Conveyor21     240        20      941   4.746719   0.310185         5         0    1.55  (Rep 12)
Conveyor21     240        20      926   5.973476   0.384128         7         1    1.92  (Rep 13)
Conveyor21     240        20      948   5.668719   0.373191         7         0    1.87  (Rep 14)
Conveyor21     240        20      982   7.346960   0.501022        11         1    2.51  (Rep 15)
Conveyor21     240        20    968.2   6.141139   0.413491   7.26667  0.266667    2.07  (Average)
Conveyor21      0          0  23.2108   0.748129  0.0561181   1.57963  0.457738    0.28(Std. Dev.)

LOCATION STATES BY PERCENTAGE (Single Capacity/Tanks)
Location  Scheduled          %      %      %        %        %     %
Name          Hours  Operation  Setup   Idle  Waiting  Blocked  Down
--------  ---------  ---------  -----  -----  -------  -------  ----
Lathe           240      42.25   0.00  57.75     0.00     0.00  0.00  (Rep 1)
Lathe           240      40.64   0.00  59.36     0.00     0.00  0.00  (Rep 2)
Lathe           240      38.27   0.00  61.73     0.00     0.00  0.00  (Rep 3)
Lathe           240      40.41   0.00  59.59     0.00     0.00  0.00  (Rep 4)
Lathe           240      40.43   0.00  59.57     0.00     0.00  0.00  (Rep 5)
Lathe           240      40.89   0.00  59.11     0.00     0.00  0.00  (Rep 6)
Lathe           240      39.83   0.00  60.17     0.00     0.00  0.00  (Rep 7)
Lathe           240      42.06   0.00  57.94     0.00     0.00  0.00  (Rep 8)
```

Lathe	240	40.99	0.00	59.01	0.00	0.00	0.00	(Rep 9)
Lathe	240	41.24	0.00	58.76	0.00	0.00	0.00	(Rep 10)
Lathe	240	39.68	0.00	60.32	0.00	0.00	0.00	(Rep 11)
Lathe	240	37.68	0.00	62.32	0.00	0.00	0.00	(Rep 12)
Lathe	240	39.83	0.00	60.17	0.00	0.00	0.00	(Rep 13)
Lathe	240	38.91	0.00	61.09	0.00	0.00	0.00	(Rep 14)
Lathe	240	40.13	0.00	59.87	0.00	0.00	0.00	(Rep 15)
Lathe	240	40.22	0.00	59.78	0.00	0.00	0.00	(Average)
Lathe	0	1.27	0.00	1.27	0.00	0.00	0.00	(Std. Dev.)

RESOURCES

Resource Name	Units	Scheduled Hours	Number Of Times Used	Average Minutes Per Usage	Average Minutes Travel To Use	Average Minutes Travel To Park	% Blocked In Travel	% Util	
Gryphon1	1	240	1999	0.373000	0.239880	0.000000	0.00	8.51	(Rep 1)
Gryphon1	1	240	1946	0.373000	0.239753	0.000000	0.00	8.28	(Rep 2)
Gryphon1	1	240	1922	0.373000	0.239750	0.000000	0.00	8.18	(Rep 3)
Gryphon1	1	240	1938	0.373000	0.239752	0.000000	0.00	8.25	(Rep 4)
Gryphon1	1	240	1909	0.373000	0.239874	0.000000	0.00	8.12	(Rep 5)
Gryphon1	1	240	1964	0.373000	0.239756	0.000000	0.00	8.36	(Rep 6)
Gryphon1	1	240	1877	0.373000	0.239872	0.000000	0.00	7.99	(Rep 7)
Gryphon1	1	240	1987	0.373000	0.239879	0.000000	0.00	8.46	(Rep 8)
Gryphon1	1	240	1927	0.373000	0.239875	0.000000	0.00	8.20	(Rep 9)
Gryphon1	1	240	2008	0.373000	0.239761	0.000000	0.00	8.54	(Rep 10)
Gryphon1	1	240	1965	0.372818	0.239878	0.000000	0.00	8.36	(Rep 11)
Gryphon1	1	240	1882	0.373000	0.239745	0.000000	0.00	8.01	(Rep 12)
Gryphon1	1	240	1850	0.373000	0.239754	0.000000	0.00	7.87	(Rep 13)
Gryphon1	1	240	1896	0.373000	0.239747	0.000000	0.00	8.07	(Rep 14)
Gryphon1	1	240	1962	0.373000	0.239816	0.000000	0.00	8.35	(Rep 15)
Gryphon1	1	240	1935.47	0.372988	0.239806	0.000000	0.00	8.24	(Average)
Gryphon1	0	0	46.7163	0.000047	0.000062	0.000000	0.00	0.20	(Std. Dev.)

ENTITY ACTIVITY

Entity Name	Total Exits	Current Quantity In System	Average Minutes In System	Average Minutes In Move Logic	Average Minutes Wait For Res, etc.	Average Minutes In Operation	Average Minutes Blocked	
PartA	999	1	13.380452	1.411779	2.670270	6.085443	3.212960	(Rep 1)
PartA	973	0	13.283904	1.399292	3.063450	6.016173	2.804990	(Rep 2)
PartA	961	0	11.754903	1.395550	2.006039	5.734906	2.618408	(Rep 3)
PartA	969	0	12.991538	1.402801	2.668086	6.005700	2.914951	(Rep 4)
PartA	954	1	12.266941	1.397845	2.083270	6.102501	2.683325	(Rep 5)
PartA	982	0	14.156283	1.412184	3.748663	5.997648	2.997788	(Rep 6)
PartA	938	2	13.168057	1.406469	2.533362	6.107623	3.120602	(Rep 7)
PartA	993	1	12.349664	1.413047	1.919710	6.095514	2.921393	(Rep 8)
PartA	963	2	12.912779	1.407113	2.308750	6.128580	3.068335	(Rep 9)
PartA	1004	0	13.523286	1.408617	3.165776	5.915837	3.033057	(Rep 10)
PartA	982	1	12.841741	1.408166	2.773991	5.820079	2.839505	(Rep 11)
PartA	941	0	11.260368	1.385254	1.586367	5.767648	2.521099	(Rep 12)
PartA	925	1	12.927238	1.395378	2.424010	6.201590	2.906259	(Rep 13)
PartA	948	0	12.325734	1.397912	2.221241	5.911015	2.795566	(Rep 14)
PartA	981	1	13.991702	1.408088	3.811506	5.891619	2.880490	(Rep 15)
PartA	967.533	0.666667	12.875639	1.403300	2.598966	5.985458	2.887915	(Average)
PartA	23.3203	0.723747	0.785832	0.007878	0.640069	0.140457	0.187945	(Std. Dev.)

VARIABLES

Variable Name	Total Changes	Average Minutes Per Change	Minimum Value	Maximum Value	Current Value	Average Value	
Total Parts Num A	1000	14.393448	0	1000	1000	490.209	(Rep 1)
Total Parts Num A	973	14.793526	0	973	973	481.538	(Rep 2)
Total Parts Num A	961	14.961375	0	961	961	470.511	(Rep 3)
Total Parts Num A	969	14.831268	0	969	969	480.712	(Rep 4)
Total Parts Num A	955	15.070408	0	955	955	471.198	(Rep 5)
Total Parts Num A	982	14.634240	0	982	982	499.783	(Rep 6)
Total Parts Num A	940	15.316354	0	940	940	468.207	(Rep 7)
Total Parts Num A	994	14.480016	0	994	994	499.416	(Rep 8)
Total Parts Num A	965	14.920545	0	965	965	482.179	(Rep 9)

Total Parts Num A	1004	14.333513	0	1004	1004	495.754	(Rep 10)
Total Parts Num A	983	14.648528	0	983	983	503.422	(Rep 11)
Total Parts Num A	941	15.287030	0	941	941	478.249	(Rep 12)
Total Parts Num A	926	15.550469	0	926	926	475.7	(Rep 13)
Total Parts Num A	948	15.176036	0	948	948	460.533	(Rep 14)
Total Parts Num A	982	14.663585	0	982	982	489.122	(Rep 15)
Total Parts Num A	968.2	14.870689	0	968.2	968.2	483.102	(Average)
Total Parts Num A	23.2108	0.358817	0	23.2108	23.2108	12.8813	(Std. Dev.)

C.1.2 Assembly Programs

C.1.2.1 Code running on DSP 1

```
*-----------------------------------------------------------------------
* 1MHard1.ASM -- Hardware-based simulation,
* of a scenario: 1 machine, 1 part type, 1 robot, and no machine breakdown
* This is the program running on Microprocessor 1, using to simulate arrival event
* Programmed by: Dong Xu
* Date         : Nov. 11, 2000
* Modified     : Mar. 2, 2001
*-----------------------------------------------------------------------
        .start "M1data", 0xB00000   ;M1 random data section, B00000-B03FFF
        .start "M2data", 0xB04000   ;M2 random data section, B04000-B07FFF
        .start "data", 0x809900     ;data section, 809900-809EFF
        .start "shared", 0xA00000   ;Shared memory section, A00000-A003FF
        .start "code", 0x809B00     ;code section
        .start "isrcode1", 0x809C50 ;Interrupt service routine 1
        .start "isrcode2", 0x809D00 ;Interrupt service routine 2

        .sect "M1data"
;M1 random number
RandNum1    .float 0.0  ;first of generated random number

        .sect "M2data"
;M2 random number
RandNum2    .float 0.0  ;first of generated random number

        .sect "data"
;Constants for Timer 0 control
EINT    .word 00002000h    ;Enable global interrupt
ETINT0  .word 00000100h    ;Enable Timer0 interrupt
PERIOD  .int 100000        ;Timer period
BISR1   .word 60809C50h    ;Branch to interrupt service routine, 60 jump to 0x809D00
BISR2   .word 60809D00h    ;Jump to 0x809D00
COMPFIN .int 0             ;Computation finished
TimersPort .word 00D00000h
HALT    .int 1

        .sect "shared"
COUNT   .int 0             ;Count of Timer interrupt
TOTCYC  .int 0             ;Total cycles
SysExit .int 5             ;Exit state of the simulation,
;                            0 success, 1 event list empty, 2 Lathe 1 input queue overflow
;                            3 Mill 1 input queue overflow, 5 other fatal error, 6 ln negative

;These global variables used on M1 to pass parameters between functions to execute the simulation
;Average of simulation results
M1AveInQ      .float 0.0  ;Average of number parts in queue 1
M1AveM1       .float 0.0  ;Average of machine 1 utilization
M1AveBlock    .float 0.0  ;Average block time of parts on machine 1
M1AveR        .float 0.0  ;Average of robot 1 utilization
M1AveStay     .float 0.0  ;Average time stay on machine 1
M1AveDelay    .float 0.0  ;Average delay time of parts on machine 1

M1NumInQ      .int 0      ;Number of parts waiting in M1 input queue
M1TotNum      .int 1      ;Total number of part entering
M1SumPartA    .int 0      ;Total finished Part A
M1PState      .int 0      ;Previous state of machine 1 before broken

M1SumInQ      .float 0.0  ;Sum of number parts in queue
M1SumM1       .float 0.0  ;Sum of machine 1 utilization
M1SumDelay    .float 0.0  ;Total delay time of parts
M1SumBlock    .float 0.0  ;Total block time of parts
M1SumStay     .float 0.0  ;Total time stay in system
M1MeArriA     .float 15.0 ;Mean time between arrival of Part A

Time        .float 0.0    ;Current simulating time, in minutes
LastTime    .float 0.0    ;Time of last event
EndTime     .float 14400.0;End simulation time, 240 hours
```

```
NxtEvet     .int 0        ;Next event type, 0=empty, 1=arrival on M1, 2=departure on M1,
3=machine 1 broken
                          ;4=Part B arrival on M2, 5 = Part A arrival on M2, 6 = departure on M2

;Event list
;current event
EveList0    .int 0      ;Event type = 0(current), 1(Arrival), 2(Departure), 3(Machine Broken)
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 1      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
;Part A arrival event on M1
EveList1    .int 1      ;Event type = 0(current), 1(Arrival), 2(Departure), 3(Machine Broken)
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 1      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
;Part A departure event on M1
EveList2    .int 2      ;Event type = 0(`current), 1(Arrival), 2(Departure), 3(Machine Broken)
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 1      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time

M1InputQ0   .space 312  ;Input queue for the workstation 1, =6*51+6 for tail
;Robot path network, should use travel time if necessary
RNetwork0   .float 0.0, 0.0, 0.0, 0.0
            .float 0.0, 0.0, 0.3, 0.6
            .float 0.0, 0.3, 0.0, 0.3
            .float 0.0, 0.6, 0.3, 0.0

;Define the pointers to the arrays
EveList     .word EveList0
M1InputQ    .word M1InputQ0
RNetwork    .word RNetwork0
M1PCurPart  .word M1CurPart
PM1         .word M1

;Defined the structure used in the program
;Describe workstation 1
M1          .int 1      ;Machine type, 1 = lathe, 2 = mill
            .int 0      ;Machine state = 0(idle), 1(busy), 2(blocked), 3(broken)
            .int 0      ;Current operating part type, 0 = Part A, 1 = Part B
            .float 6.0  ;Processing time for Part A
            .float 0.0  ;Processing time for Part B, not available
            .float 450  ;Mean time between failure
            .float 25   ;Mean time to repair

;Current part scheduled to be processed on machine
M1CurPart   .int 0      ;Part type = 0(Part A), 1(Part B)
            .int 1      ;Operation required on workstation 1, 0(No), 1(Yes)
            .int 1      ;Operation required on workstation 2, 0(No), 1(Yes)
            .float 0.0  ;Time when entering the FMS
            .float 0.0  ;Processing start time to one workstation
            .float 0.0  ;Finish time of one operation on one workstation

;Describe the robot
Grephon1    .int 0      ;Robot state = 0(idle), 1(occupied)
            .int 1      ;current position of robot
            .float 0.0  ;Next available time of the robot
            .float 0.0  ;Area of robot utilization

*-------------------------------------------------------------------------
*Define references
*-------------------------------------------------------------------------
M1Counter   .int 0              ;Counter of M1 random number generation
M2Counter   .int 0              ;Counter of M2 random number generation
RandTotNum  .word 0x00003FFE    ;Length limit of the data
BigRandNum  .float -15          ;Random number is unusually small

PRandNum1   .word RandNum1
PRandNum2   .word RandNum2

*-------------------------------------------------------------------------
```

```
* DEFINE CONSTANTS
*-------------------------------------------------------------------------
QLimit      .int 50             ;Length limit of the queue
NumEvet     .int 2              ;Number of event
NumPType    .int 1              ;Number of part type

;Constant used by random number generation subroutine
A       .int    1078373         ; Constants needed for RAND
C       .int    2311527         ;
M1SEED  .int    0               ;
M2SEED  .int    0
MASK    .word   0xFF7FFFFF      ;Mask for fast inverse float
MASK1   .word   0x007FFFFF      ;Mask for exponential distribution, %2^23
INVE23  .word   0xE9000000      ;1/2^23
BigNum  .float  1.0e29          ;Constant big value
        .float  1.0e30          ;

;Internal constants for ln(u)
;Scaling coefficients for ln(1+x)
LNRM    .float  0.6931471806    ; LN(2)
C0      .float  1.0000000000    ; C0 (1.0)
;       Polynomial coefficients for ln(1+X), 0 <= X < 1.
        .float  0.9999964239    ; TOP OF C1
        .float  -0.4998741238   ; TOP OF C2
        .float  0.3317990258    ; TOP OF C3
        .float  -0.2407338084   ; TOP OF C4
        .float  0.1676540711    ; TOP OF C5
        .float  -0.0953293897   ; TOP OF C6
        .float  0.0360884937    ; TOP OF C7
C8      .float  -0.0064535442   ; TOP OF C8
AC8     .word   C8

*-------------------------------------------------------------------------------
*Special signals for synchronization purpose, should be stored in the shared memory,
*accessed by both Microprocessor
*-------------------------------------------------------------------------------
M2Finish    .int 1      ;Indicate M2 finish the part departure event
M2SafeTime  .float -1.0 ;Indicate how far simulation clock on M2 can advance
M2D_Time    .float 0.0  ;Scheduled next part departure time on microprocessor 2

SharedMem   .word $+1

    .sect "code"
    .entry  _main
*-------------------------------------------------------------------------
* FUNCTION DEF : _main
* Control the advance of the program
*-------------------------------------------------------------------------
_main:
        LDP     @_main          ;Set Data Segment Pointer
        LDI     @STACK,SP       ;Set stack pointer
        LDP     @SharedMem
        CALL    _M1RandNum      ;Prepare random number
        LDP     @_main

        LDP     @SharedMem
        LDI     @PRandNum1,AR3  ;Index of random data series
;Setup Timer 0 for recording execution time
        LDI     @BISR2, R0
        STI     R0, @9FC9h      ;Save the branch instruction to Timer0 interrupt vector
        LDI     200h, R1
        STI     R1, @8020h      ;Using internal H1/2 clock for Timer 0, halt timer 0
        LDI     @PERIOD, R1
        STI     R1, @8028h      ;Set period of Timer 0
        LDI     0,R1
        STI     R1,@8024h       ;Reset counter register
        LDI     2C0h, R1
        STI     R1, @8020h      ;Start Timer 0
        OR      @ETINT0, IE     ;Enable Timer 0 interrupt
        OR      @EINT, ST       ;Enable Interrupt
```

```
        CALL    _initial        ;Initialize the simulation, especially data structure

LM1     LDF     @Time,R0
        CMPF    @EndTime,R0     ;Does simulation end?
        BGT     LM2

        CALL    _timing         ;Determine the next event type

;Invoke the appropriate event handle function
        NOP
        B       LM1
LM2
        CALL    _M1report       ;Report simulation results

        LDI     0,R0
        STI     R0,@SysExit     ;Successfully run

;Calculate the execution time
        LDP     @_main
        LDI     200h, R1
        STI     R1, @8020h      ;Halt the timer
        LDI     @PERIOD, R0     ;Calculate the whole cycles
        LDI     @8024h, R1      ;Current counter number
        LDP     @SharedMem
        MPYI    @COUNT, R0
        ADDI    R1, R0
        STI     R0, @TOTCYC     ;Total cycle used
        LDF     @BigNum+1,R0
        STF     R0,@M2SafeTime  ;Inform M2 to finish its processing
        LDP     @_main
END     B       $               ;End of the program


*-------------------------------------------------------------------------
* FUNCTION DEF : _M1RandNum
* Fill the random data
* Input        : None
* Register     : R0,R1,AR5
* Output       : None
*-------------------------------------------------------------------------
_M1RandNum:
        LDI     @PRandNum1, AR5 ;AR5 is the index for storage of M1 random numbers
        LDI     0,R0
        STI     R0,@M1Counter   ;Initialize counter

LR1     LDI     @M1Counter,R0
        CMPI    @RandTotNum,R0  ;Does process end?
        BGE     LR4

;Increase counter
        LDI     @M1Counter,R0
        ADDI    1,R0
        STI     R0,@M1Counter
;Generate random number for M1
        CALL    _M1lnUni         ;
        STF     R0, *AR5++(1)    ;Store the random number
        LDF     *-AR5(1),R2      ;Read the stored random number
        PUSHF   R0
        POPF    R0               ;Set R0 to the 32-bit
        CMPF    R2,R0            ;
        BZ      LR1              ;If same, continue
        LDI     11, R1
        STI     R1, @SysExit     ;Indicate M1 read/write wrong
        CALL    _exit

LR4     LDF     0.0,R0
        STF     R0,*AR5          ;End of data
        LDI     @M1Counter,R0
        STI     R0,*+AR5(1)      ;Total number of data
        LDI     @PRandNum1,AR5
        LDI     0,R1             ;Index
```

```
        STI     R1,@M1Counter
LR5     LDF     *AR5++(1),R0,
        BP      LR6             ;Positive?
        CMPF    @BigRandNum,R0
        BLT     LR7             ;Too small?
        LDI     @M1Counter,R1
        ADDI    1,R1
        STI     R1,@M1Counter
        CMPI    @RandTotNum,R1
        BLT     LR5

        LDI     10,R0
        STI     R0,@SysExit     ;Successfully run
        RETS
LR6     LDI     14,R0
        STI     R0,@SysExit     ;positive number found
        CALL    _exit
LR7     LDI     15,R0
        STI     R0,@SysExit     ;small number found
        CALL    _exit


*-------------------------------------------------------------------------
* FUNCTION DEF : _initial
* Fill the robot path network, initialize the data structures and the first arrival
* Input        : None
* Register     : R1, AR0, RC, RS, RE
* Output       : None
*-------------------------------------------------------------------------
_initial:
;Describe the current part on workstation 1, InputQ[0] is the storage of next part
        LDI     0,R1
        STI     R1,@M1InputQ0     ;Part A
        LDI     1,R1
        STI     R1,@M1InputQ0+1   ;Part A need turning operation on workstation 1
        STI     R1,@M1InputQ0+2   ;Part A need drilling operation on workstation 2
        LDF     @M1MeArriA,R0
        CALL    _M1expon
        STF     R0,@M1InputQ0+3   ;Part A first arrival time
        LDF     0.0,R1
        STF     R1,@M1InputQ0+4   ;Part A start on workstation 1, depending on robot
        STF     R1,@M1InputQ0+5   ;Time to finish operation on workstation 1 does not known yet

        STF     R0,@EveList1+3    ;Schedule the first arrive time
        LDF     @BigNum+1,R1
        STF     R1,@EveList2+3    ;Schedule the initial departure time to be infinite
        LDI     @TimersPort,AR7   ;
;Setup Timer 1 for arrival event
        LDI     @BISR1, R0
        STI     R0, @9FCAh      ;Save the branch instruction to Timer1 interrupt vector
        LDI     200h, R1
        STI     R1, @8030h      ;Using internal H1/2 clock for Timer 1, halt timer 1
        LDF     @EveList1+3, R1
        FIX     R1,R1
        MPYI    10,R1           ;Time scaling at 10
        STI     R1, @8038h      ;Set period of Timer 1
        LDI     0,R1
        STI     R1,@8034h       ;Reset counter register
        LDI     2C0h, R1
        STI     R1, @8030h      ;Start Timer 1
        RETS

*-------------------------------------------------------------------------
* FUNCTION DEF : _timing
* schedule the next event, synchronize and advance the simulation clock
* Input        : None
* Register     : R0, R1, R2, AR0, DP
* Output       : None
*-------------------------------------------------------------------------
_timing:
LT1     LDI     @COMPFIN,R1
```

```
        BZ      LT2
        LDI     200h, R1
        STI     R1, @8030h      ;Using internal H1/2 clock for Timer 1, halt timer 1
        LDF     @EveList1+3, R1
        FIX     R1,R1
        MPYI    10,R1           ;Time scaling at 10
        STI     R1, @8038h      ;Set period of Timer 1
        LDI     0,R1
        STI     R1,@8034h       ;Reset counter register
        LDI     2C0h, R1
        STI     R1, @8030h      ;Start Timer 1
        LDF     @EveList1+3,R1
        STF     R1,@M2SafeTime  ;M2 safe time

        STF     R1,@Time        ;Update Simulation clock
        LDF     @LastTime,R0    ;Last event time
        SUBF3   R0,R1,R2        ;Compute time since last event
        STF     R1, @LastTime   ;Save last event time

        FLOAT   @M1NumInQ,R0
        MPYF    R2,R0
        ADDF    @M1SumInQ,R0
        STF     R0,@M1SumInQ    ;Update area of number in M1 queue
        B       LTEND
LT2     LDI     200h, R1
        STI     R1, @8030h      ;Halt timers
        LDI     0,R1
        STI     R1,@HALT
        STI     R1,*AR7
LTEND   RETS

*-------------------------------------------------------------------------
* FUNCTION DEF : _M1rerobot
* handle robot request, return robot 1 travel time.
* Input        : R0,initial position, R1, destination position
* Register     : R0, R1, AR0, IR0, IR1, (only AR0,IR0,IR1 restored)
* Output       : R0 travel time
*-------------------------------------------------------------------------
_M1rerobot:
;Calculate travel time, robot need to travel from current position to initial position,
;then from initial position to the destination
        PUSH    AR0
        PUSH    IR0
        PUSH    IR1

        LDI     @Grephon1+1,IR0
        MPYI    4,IR0
        ADDI    R0,IR0                  ;From current position to starting position
        LDI     @RNetwork,AR0
        LDI     R0,IR1
        MPYI    4,IR1
        ADDI    R1,IR1                  ;From starting position to destination
;Robot traveltime = RNetwork[Grephon.R_Pos][initpos] + RNetwork[initpos][destpos]
        ADDF3   *+AR0(IR1),*+AR0(IR0),R0
        STI     R1,@Grephon1+1           ;Next robot position
        LDF     @Time,R1
        CMPF    @Grephon1+2,R1
        BGE     M1LR1
        ADDF    @Grephon1+2,R0           ;Robot is occupied, considering some waiting time
        SUBF    R1,R0
;Next time robot stay at initial position and available time is current time plus travel time
M1LR1   ADDF    R0,R1                   ;Next available time
        STF     R1,@Grephon1+2
        LDF     2.0,R1
        MPYF    R0,R1
        ADDF    @Grephon1+3,R1           ;Update robot utilization
        STF     R1,@Grephon1+3

        POP     IR1
        POP     IR0
        POP     AR0
```

```
        RETS

*---------------------------------------------------------------------
* FUNCTION DEF : _M1report
* report simulation results
* Input        : None
* Register     : R0, R2, R5
* Output       : None
*---------------------------------------------------------------------
_M1report:
        LDF     @EndTime,R0
        CALL    INV_F30
        RND     R0,R5           ;R5=1/Time

        LDF     @M1SumM1,R0
        MPYF    R5,R0
        STF     R0,@M1AveM1     ;Utilization of M1

        LDF     @Grephon1+3,R0
        MPYF    R5,R0
        STF     R0,@M1AveR      ;Utilization of robot

        LDF     @M1SumBlock,R0
        MPYF    R5,R0
        STF     R0,@M1AveBlock    ;Average blocked

        LDF     @M1SumInQ,R0
        MPYF    R5,R0
        STF     R0,@M1AveInQ      ;Average waiting in queue

        FLOAT   @M1SumPartA,R0
        CALL    INV_F30
        RND     R0,R2           ;R2 = 1/SumPart
        LDF     @M1SumDelay,R0
        MPYF    R2,R0
        STF     R0,@M1AveDelay    ;Average delay

        LDF     @M1SumStay,R0
        MPYF    R2,R0
        STF     R0,@M1AveStay     ;Average stay in FMS
        RETS

*---------------------------------------------------------------------
* FUNCTION DEF : _M1expon
* Generate a random number with exponential distribution
* Input        : R0 is the mean
* Register     : R1
* Output       : R0 is the random number with exponential distribution
*---------------------------------------------------------------------
_M1expon:
*Fast 32 bit uniform random number generator
        PUSH    R1
        PUSHF   R1

        NEGF    R0, R1          ; R1 is the negative of the mean
        PUSH    R1
        PUSHF   R1              ; Save -mean in the stack, all 40 bits
        LDF     *AR3++(1),R0    ; Get a random number value
        POPF    R1              ; -Mean
        POP     R1
        MPYF    R1, R0          ; random number =  -mean*ln(u)

        POPF    R1
        POP     R1
        RETS

*---------------------------------------------------------------------
* FUNCTION DEF : _M1lnUni
* Generate a random number with exponential distribution
* Input        : None
```

```
* Register     : R0
* Output       : R0 Is the ln(uniform) random number series
*-----------------------------------------------------------------------
_M1lnUni:
        LDI     @M1SEED,R0      ; Call here for last SEED
        MPYI    @A,R0           ; Calculate r = (A*r+C)%m
        ADDI    @C,R0           ;
        AND     @MASK1, R0      ; m=2^23
        STI     R0,@M1SEED      ; Result is returned
        ADDI    1,R0            ;
        FLOAT   R0,R0           ;
        MPYF    @INVE23,R0      ; r/m
        CALL    _ln             ; ln(u)
        RETS

*-------------------------------------------------------------------------
*   Function: _ln
*   Logarithm function base E: R0 = LN(R0)
*   Input   : R0 > 0.0
*   Register: R0-3, AR0, DP
*   Register altered: R0-3
*   Output  : R0
*   Cycles  : 57
*-------------------------------------------------------------------------
_ln:
        PUSH    R1
        PUSHF   R1
        PUSH    R2
        PUSHF   R2
        PUSH    R3
        PUSHF   R3
        PUSH    AR0             ; Save AR0
        PUSH    RC
        PUSH    RS
        PUSH    RE              ; Save the repeat counters

        CMPF    0.0,R0          ; Test X < 0.0
        BGT     LN1
        LDI     6,R1
        STI     R1,@SysExit     ; 6 means float error
        CALL    _exit           ; Return if X <= 0

;       Scale variable X
LN1     PUSH    DP              ; Save DP
        LDP     @AC8            ; Load data page pointer
        PUSHF   R0              ; Save as floating point number
        POP     R3              ; R3 in integer format to check exp.
        ASH     -24,R3          ; R3 holds singed exp.
        FLOAT   R3,R1           ; R1 = singed float number of exp.
        LDF     @C0,R2          ; R2 = 1.0
        LDE     R2,R0           ; Exp. of R0 set to 0 (1 <= X < 2)
        SUBRF   R0,R2           ; R2 = X - 1 (0 <= X < 1)
        LDF     @LNRM,R0        ; R0 = LN(2)
        MPYF    R1,R0           ; R0 = Exp.*LN(2)
        LDF     R0,R3           ; R3 = Exp.*LN(2)
        LDI     @AC8,AR0        ; AR0 point to coefficient table
        POP     DP              ; UNSAVE DP

;       EVALUATE TRUNCATED SERIES

        RND     R2,R1           ; R1 = RND X
        MPYF3   *AR0--,R1,R0    ; R0 = X*C8
        ADDF    *AR0--,R0       ; R0 = C7 + R0

        LDI     5, RC           ; Repeat 6 times
        RPTB    NR_LN
        MPYF    R1,R0           ; R0 = X*(C(i) + R0)
NR_LN   ADDF    *AR0--,R0       ; R0 = C(i-1) + R0

;       ADD IN SCALED EXPONENT AND RETURN
        RND     R0,R0           ; Round before multiply
```

```
        MPYF    R1,R0           ; R0 = X*(C1 + R0)
        ADDF    R3,R0           ; R0 = LN(X) + E*LN(2)

        POP     RE
        POP     RS
        POP     RC              ; Restore repeat counters
        POP     AR0             ; Restore AR0
        POPF    R3
        POP     R3
        POPF    R2
        POP     R2
        POPF    R1
        POP     R1
        RETS

*-------------------------------------------------------------------------
* FUNCTION DEF : INV_F30
* R0=INVF(R0)
*   Calculates the inverse float of the value of R0 and
*   returns the result in R0.
* Input        : R0
* Registers    : R1, R2, RC
* Output       : R0 = 1/R0
* Cycles       : 42
*-------------------------------------------------------------------------
INV_F30:
        PUSH    R1              ;
        PUSHF   R1              ; Save all 40 bits of R1
        PUSH    R2              ;
        PUSHF   R2              ; Save all 40 bits of R2
        PUSH    RC
        PUSH    RS
        PUSH    RE              ; Save the repeat counters

        LDF     R0,R1           ;
        PUSHF   R1              ; Access R0's bit fields as integers
        POP     R1              ;
        XOR     @MASK,R1        ; invert bits. except sign, eq to log inverse
        PUSH    R1              ; put back as float
        POPF    R1              ;
        ;-----------------------------------------
        ; Now iterate Newton Raphson reduction 5 times
        ;-----------------------------------------
        LDI     4,RC            ; Set up a block repeat (expand for speed!)
        RPTB    NR_INV          ;
        MPYF3   R1,R0,R2        ; R0 = v * x[0]
        SUBRF   2.0,R2          ; R0 = 2.0 - v * x[0]
NR_INV  MPYF    R2,R1           ; R2 = x[1] = x[0] * (2.0 - v * x[0])
        LDF     R1,R0

        POP     RE
        POP     RS
        POP     RC              ; Restore repeat counters
        POPF    R2              ; Restore exp. of R2
        POP     R2              ; Restore man. of R2
        POPF    R1              ; Restore exp. of R1
        POP     R1              ; Restore man. of R1
        RETS

*-------------------------------------------------------------------------
* FUNCTION DEF : _exit
* Input        : none
* Register     : none
* Output       : none
*-------------------------------------------------------------------------
_exit:
EXITSIM B       $

        .sect "isrcode1"
*Interrupt Service Routine
*-------------------------------------------------------------------------
```

```
* FUNCTION DEF : _M1arrive
* schedule the next arrival event and handle current arrival event on M1
* Input        : None
* Register     : R0, R1, R2, AR0, AR1
* Output       : None
*-----------------------------------------------------------------------
_M1arrive:
        LDI     0,R0
        STI     R0,@COMPFIN     ;Computation not finished
        LDI     @M1+1,R0        ;Check to see if workstation 1 is idle?
        BNZ     M1LA1           ;If not idle, go to LA1
        LDI     1,R0            ;If idle
        STI     R0,@M1+1        ;Set Machine 1 busy
        LDI     2,R1            ;Request robot, Load part from queue to machine 1
        CALL    _M1rerobot      ;R0 = initial position, R1 = destination position
                                ;Travel time return in R0
        LDF     @Time,R1
        STF     R1,@M1CurPart+3 ;Current part entering time
        ADDF3   R0,R1,R2
        STF     R2,@M1CurPart+4 ;Current process starting time
        LDF     @M1+3,R0
        CALL    _M1expon        ;Processing time for the part
        ADDF    @M1CurPart+4,R0
        STF     R0,@EveList2+3  ;Schedule a Part 1 departure after processing
                                ;considering the travel time of robot
        STF     R0,@M1CurPart+5 ;Current part finishing time
        BRD     M1LA4
        ADDF    @M1SumM1,R0
        SUBF    @M1CurPart+4,R0
        STF     R0,@M1SumM1     ;Update Machine busy period
*** B   M1LA4 ;BRANCH OCCURS
;If machine is not available (either busy, blocked or broken), put the current part in queue
M1LA1   LDI     @M1NumInQ,R1
        ADDI    1,R1
        STI     R1,@M1NumInQ    ;Increase the length of queue
        CMPI    @QLimit,R1      ;Check if queue is overflow?
        BLE     M1LA2
        LDI     2,R0            ;Queue is overflow, exit 2
        STI     R0,@SysExit
        CALL    _exit
M1LA2   LDI     @M1NumInQ,AR0   ;InputQ[NumInQ] = InputQ[0]
        MPYI    6,AR0
        ADDI    @M1InputQ,AR0
        LDI     @M1InputQ,AR1
        LDI     5,RC
        RPTB    M1LA3
        LDI     *AR1++,R0       ;
M1LA3   STI     R0,*AR0++
;Schedule next Part A arrival
M1LA4   LDF     @M1MeArriA,R0
        CALL    _M1expon
        ADDF    @Time,R0
        STF     R0,@M1InputQ0+3
        STF     R0,@EveList1+3  ;Next arrival time
        LDI     @M1TotNum,R1    ;Update total entering part number
        ADDI    1,R1
        STI     R1,@M1TotNum
        LDI     1,R0
        STI     R0,@COMPFIN     ;Computation finished
        LDI     @HALT,R0
        BNZ     LAEND
        LDI     2C0h, R1
        STI     R1, @8030h      ;Start Timer 1
        LDI     1,R0
        STI     R0,@HALT
        STI     R0,*AR7
LAEND   RETI

        .sect "isrcode2"
*Interrupt Service Routine
_ISRCount:
```

```
        PUSH R0
        PUSHF R0
        LDI @COUNT, R0
        ADDI 1, R0
        STI R0, @COUNT
        POPF R0
        POP R0
        RETI

STACK   .word  $+1          ; Bottom of the Stack
        .end
```

C.1.2.2 Code running on DSP 2

```
*-----------------------------------------------------------------------
* 1MHard2.ASM -- Hardware-based simulation
* of a scenario: 1 machine, 1 part type, 1 robot, and no machine breakdown
* This is the program running on Microprocessor 2, using to simulate departure event
* Programmed by: Dong Xu
* Date         : August 8, 2000
* Modified     : Feb. 25, 2001
*-----------------------------------------------------------------------
        .start "M1data", 0xB00000   ;M1 random data section, B00000-B03FFF
        .start "M2data", 0xB04000   ;M2 random data section, B04000-B07FFF
        .start "data", 0x809900    ;data section, 809900-809EFF
        .start "shared", 0xA00000   ;Shared memory section, A00000-A003FF
        .start "code", 0x809B00     ;code section
        .start "isrcode1", 0x809C50 ;Interrupt service routine 1

        .sect "M1data"
;M1 random number
RandNum1    .float 0.0  ;first of generated random number

        .sect "M2data"
;M2 random number
RandNum2    .float 0.0  ;first of generated random number

        .sect "data"
;Constants for Timer 0 control
EINT    .word 00002000h    ;Enable global interrupt
ETINT0  .word 00000100h    ;Enable Timer0 interrupt
PERIOD  .int 100000        ;Timer period
BISR1   .word 60809C50h    ;Branch to interrupt service routine, 60 jump to 0x809C50
BISR2   .word 60809D00h    ;Jump to 0x809D00
COMPFIN .int 0             ;Computation finished
TimersPort .word 00D00000h
HALT    .int 1

        .sect "shared"
COUNT   .int 0             ;Count of Timer interrupt
TOTCYC  .int 0             ;Total cycles
SysExit .int 5             ;Exit state of the simulation,
;                            0 success, 1 event list empty, 2 Lathe 1 input queue overflow
;                            3 Mill 1 input queue overflow, 5 other fatal error, 6 ln negative

;These global variables used on M1 to pass parameters between functions to execute the simulation
;Average of simulation results
M1AveInQ      .float 0.0  ;Average of number parts in queue 1
M1AveM1       .float 0.0  ;Average of machine 1 utilization
M1AveBlock    .float 0.0  ;Average block time of parts on machine 1
M1AveR        .float 0.0  ;Average of robot 1 utilization
M1AveStay     .float 0.0  ;Average time stay on machine 1
M1AveDelay    .float 0.0  ;Average delay time of parts on machine 1

M1NumInQ      .int 0      ;Number of parts waiting in M1 input queue
M1TotNum      .int 1      ;Total number of part entering
M1SumPartA    .int 0      ;Total finished Part A
M1PState      .int 0      ;Previous state of machine 1 before broken

M1SumInQ      .float 0.0  ;Sum of number parts in queue
M1SumM1       .float 0.0  ;Sum of machine 1 utilization
M1SumDelay    .float 0.0  ;Total delay time of parts
M1SumBlock    .float 0.0  ;Total block time of parts
M1SumStay     .float 0.0  ;Total time stay in system
M1MeArriA     .float 15.0 ;Mean time between arrival of Part A

Time        .float 0.0    ;Current simulating time, in minutes
LastTime    .float 0.0    ;Time of last event
EndTime     .float 14400.0;End simulation time, 240 hours
NxtEvet     .int 0        ;Next event type, 0=empty, 1=arrival on M1, 2=departure on M1,
3=machine 1 broken
                          ;4=Part B arrival on M2, 5 = Part A arrival on M2, 6 = departure on M2

;Event list
```

```
;current event
EveList0    .int 0      ;Event type = 0(current), 1(Arrival), 2(Departure), 3(Machine Broken)
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 1      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
;Part A arrival event on M1
EveList1    .int 1      ;Event type = 0(current), 1(Arrival), 2(Departure), 3(Machine Broken)
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 1      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
;Part A departure event on M1
EveList2    .int 2      ;Event type = 0(`current), 1(Arrival), 2(Departure), 3(Machine Broken)
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 1      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time

M1InputQ0   .space 312  ;Input queue for the workstation 1, =6*51+6 for tail
;Robot path network, should use travel time if necessary
RNetwork0   .float 0.0, 0.0, 0.0, 0.0
            .float 0.0, 0.0, 0.3, 0.6
            .float 0.0, 0.3, 0.0, 0.3
            .float 0.0, 0.6, 0.3, 0.0

;Define the pointers to the arrays
EveList     .word EveList0
M1InputQ    .word M1InputQ0
RNetwork    .word RNetwork0
M1PCurPart  .word M1CurPart
PM1         .word M1

;Defined the structure used in the program
;Describe workstation 1
M1          .int 1      ;Machine type, 1 = lathe, 2 = mill
            .int 0      ;Machine state = 0(idle), 1(busy), 2(blocked), 3(broken)
            .int 0      ;Current operating part type, 0 = Part A, 1 = Part B
            .float 6.0  ;Processing time for Part A
            .float 0.0  ;Processing time for Part B, not available
            .float 450  ;Mean time between failure
            .float 25   ;Mean time to repair

;Current part scheduled to be processed on machine
M1CurPart   .int 0      ;Part type = 0(Part A), 1(Part B)
            .int 1      ;Operation required on workstation 1, 0(No), 1(Yes)
            .int 1      ;Operation required on workstation 2, 0(No), 1(Yes)
            .float 0.0  ;Time when entering the FMS
            .float 0.0  ;Processing start time to one workstation
            .float 0.0  ;Finish time of one operation on one workstation

;Describe the robot
Grephon1    .int 0      ;Robot state = 0(idle), 1(occupied)
            .int 1      ;current position of robot
            .float 0.0  ;Next available time of the robot
            .float 0.0  ;Area of robot utilization

*-----------------------------------------------------------------------
*Define references
*-----------------------------------------------------------------------
M1Counter   .int 0              ;Counter of M1 random number generation
M2Counter   .int 0              ;Counter of M2 random number generation
RandTotNum  .word 0x00003FFE    ;Length limit of the data
BigRandNum  .float -15          ;Random number is unusually small

PRandNum1   .word RandNum1
PRandNum2   .word RandNum2

*-----------------------------------------------------------------------
* DEFINE CONSTANTS
*-----------------------------------------------------------------------
QLimit      .int 50             ;Length limit of the queue
NumEvet     .int 2              ;Number of event
NumPType    .int 1              ;Number of part type
```

```
;Constant used by random number generation subroutine
A       .int    1078373          ; Constants needed for RAND
C       .int    2311527          ;
M1SEED  .int    0                ;
M2SEED  .int    0
MASK    .word   0xFF7FFFFF       ;Mask for fast inverse float
MASK1   .word   0x007FFFFF       ;Mask for exponential distribution, %2^23
INVE23  .word   0xE9000000       ;1/2^23
BigNum  .float  1.0e29           ;Constant big value
        .float  1.0e30           ;

;Internal constants for ln(u)
;Scaling coefficients for ln(1+x)
LNRM    .float  0.6931471806     ; LN(2)
C0      .float  1.0000000000     ; C0 (1.0)
;       Polynomial coefficients for ln(1+X), 0 <= X < 1.
        .float  0.9999964239     ; TOP OF C1
        .float  -0.4998741238    ; TOP OF C2
        .float  0.3317990258     ; TOP OF C3
        .float  -0.2407338084    ; TOP OF C4
        .float  0.1676540711     ; TOP OF C5
        .float  -0.0953293893    ; TOP OF C6
        .float  0.0360884937     ; TOP OF C7
C8      .float  -0.0064535442    ; TOP OF C8
AC8     .word   C8

*-------------------------------------------------------------------------------
*Special signals for synchronization purpose, should be stored in the shared memory,
*accessed by to both Microprocessor
*-------------------------------------------------------------------------------
M2Finish    .int 1      ;Indicate M2 finish the part departure event
M2SafeTime  .float -1.0 ;Indicate how far simulation clock on M2 can advance
M2D_Time    .float 0.0  ;Scheduled next part departure time on microprocessor 2

SharedMem   .word $+1


    .sect "code"
    .entry  _main
*-------------------------------------------------------------------------
* FUNCTION DEF : _main
* Control the advance of the program
*-------------------------------------------------------------------------
_main:
        LDP     @_main          ;Set Data Segment Pointer
        LDI     @STACK,SP       ;Set stack pointer
        LDP     @SharedMem
        CALL    _M2RandNum
        LDI     @PRandNum2,AR4  ;Random number data series
        CALL    _initial
        OR      @ETINT0, IE     ;Enable Timer 0 interrupt
        OR      @EINT, ST       ;Enable Interrupt

LM1     LDF     @Time,R0
        CMPF    @EndTime,R0     ;Does simulation end?
        BGT     LM2

        CALL    _timing         ;Determine the next event type
;Invoke the departure event handle function
        CALL    _M1depart       ;Handle part departure events
        B       LM1

LM2     LDF     @BigNum+1,R0
        STF     R0,@EveList2+3  ;Inform M1 to finish its processing
END     B       $               ;End of the program

*-------------------------------------------------------------------------
* FUNCTION DEF : _initial
* Fill the robot path network, initialize the data structures and the first arrival
* Input        : None
```

```
* Register     : R1, AR0, RC, RS, RE
* Output       : None
*-------------------------------------------------------------------------
_initial:
;Setup INT3 interrupt service
        LDI     @BISR1, R0
        STI     R0, @9FC4h      ;Save the branch instruction to Timer1 interrupt vector

;Setup Timer 1 for departure event
        LDI     @BISR2, R0
        STI     R0, @9FCAh      ;Save the branch instruction to Timer1 interrupt vector
        RETS
*-------------------------------------------------------------------------
* FUNCTION DEF : _M2RandNum
* Fill the robot path network, initialize the data structures and the first arrival
* Input        : None
* Register     : R0,R1,AR6
* Output       : None
*-------------------------------------------------------------------------
_M2RandNum:
        LDI     @PRandNum2, AR6 ;AR6 is the index for storage of M2 random numbers
        LDI     0,R0
        STI     R0,@M2Counter     ;Initialize counter

LR1     LDI     @M2Counter,R0
        CMPI    @RandTotNum,R0  ;Does process end?
        BGE     LR4

;Increase counter
        LDI     @M2Counter,R0
        ADDI    1,R0
        STI     R0,@M2Counter

;Generate random number for M2
LR3     CALL    _M2lnUni        ;
        STF     R0, *AR6++(1)   ;Store the random number
        LDF     *-AR6(1),R2     ;Read the stored random number
        PUSHF   R0
        POPF    R0              ;Set R0 to the 32-bit
        CMPF    R2,R0           ;
        BZ      LR1             ;If same, continue
        LDI     12, R1
        STI     R1, @SysExit    ;Indicate M2 read/write wrong
        CALL    _exit

LR4     LDF     0.0,R0
        STF     R0,*AR6         ;End of data
        LDI     @M2Counter,R0
        STI     R0,*+AR6(1)     ;Total number of data

        LDI     @PRandNum2,AR6
        LDI     0,R1            ;Index
        STI     R1,@M2Counter
LR5     LDF     *AR6++(1),R0,
        BP      LR6             ;Positive?
        CMPF    @BigRandNum,R0
        BLT     LR7             ;Too small?
        LDI     @M2Counter,R1
        ADDI    1,R1
        STI     R1,@M2Counter
        CMPI    @RandTotNum,R1
        BLT     LR5

        LDI     10,R0
        STI     R0,@SysExit     ;Successfully run
        RETS
LR6     LDI     14,R0
        STI     R0,@SysExit     ;Negative number found
        CALL    _exit
LR7     LDI     15,R0
        STI     R0,@SysExit     ;Too small number
```

```
        CALL    _exit


*-------------------------------------------------------------------------
* FUNCTION DEF : _timing
* schedule the next event, synchronize and advance the simulation clock
* Input        : None
* Register     : R0, R1, R2, AR0, DP
* Output       : None
*-------------------------------------------------------------------------
_timing:
LT1     LDI     @COMPFIN,R1
        BZ      LT2
        LDI     200h, R1
        STI     R1, @8030h      ;halt timer 1
        LDF     @EveList2+3, R1
        FIX     R1,R1
        MPYI    10,R1           ;Time scaling at 10
        STI     R1, @8038h      ;Set period of Timer 1
        LDI     0,R1
        STI     R1,@8034h       ;Reset counter register
        LDI     2C0h, R1
        STI     R1, @8030h      ;Start Timer 1
        LDF     @EveList2+3,R1  ;Departure time
        CMPF    @M2SafeTime,R1  ;Is it safe for M2 to process this departure
        BGT     LT1             ;Not safe, wait
        LDI     2,R0
        STI     R0,@NxtEvet     ;Next event type = 2

        STF     R1,@Time        ;Update Simulation clock
        LDF     @LastTime,R0    ;Last event time
        SUBF3   R0,R1,R2        ;Compute time since last event
        STF     R1, @LastTime   ;Save last event time

        FLOAT   @M1NumInQ,R0
        MPYF    R2,R0
        ADDF    @M1SumInQ,R0
        STF     R0,@M1SumInQ    ;Update area of number in M1 queue
LT2     LDI     200h, R1
        STI     R1, @8030h      ;Halt timers
        LDI     0,R1
        STI     R1,@HALT
        STI     R1,*AR7
        RETS

*-------------------------------------------------------------------------
* FUNCTION DEF : _M1rerobot
* handle robot request, return robot 1 travel time.
* Input        : R0,initial position, R1, destination position
* Register     : R0, R1, AR0, IR0, IR1, (only AR0,IR0,IR1 restored)
* Output       : R0 travel time
*-------------------------------------------------------------------------
_M1rerobot:
;Calculate travel time, robot need to travel from current position to initial position,
;then from initial position to the destination
        PUSH    AR0
        PUSH    IR0
        PUSH    IR1

        LDI     @Grephon1+1,IR0
        MPYI    4,IR0
        ADDI    R0,IR0                  ;From current position to starting position
        LDI     @RNetwork,AR0
        LDI     R0,IR1
        MPYI    4,IR1
        ADDI    R1,IR1                  ;From starting position to destination
;Robot traveltime = RNetwork[Grephon.R_Pos][initpos] + RNetwork[initpos][destpos]
        ADDF3   *+AR0(IR1),*+AR0(IR0),R0
        STI     R1,@Grephon1+1           ;Next robot position
        LDF     @Time,R1
        CMPF    @Grephon1+2,R1
```

```
        BGE     M1LR1
        ADDF    @Grephon1+2,R0          ;Robot is occupied, considering some waiting time
        SUBF    R1,R0
;Next time robot stay at initial position and available time is current time plus travel time
M1LR1   ADDF    R0,R1                   ;Next available time
        STF     R1,@Grephon1+2
        LDF     2.0,R1
        MPYF    R0,R1
        ADDF    @Grephon1+3,R1          ;Update robot utilization
        STF     R1,@Grephon1+3

        POP     IR1
        POP     IR0
        POP     AR0

        RETS

*-----------------------------------------------------------------------
* FUNCTION DEF : _M2expon
* Generate a random number with exponential distribution
* Input        : R0 is the mean
* Register     : R1
* Output       : R0 is the random number with exponential distribution
*-----------------------------------------------------------------------
_M2expon:
*Fast 32 bit uniform random number generator
        PUSH    R1
        PUSHF   R1

        NEGF    R0, R1          ; R1 is the negative of the mean
        PUSH    R1
        PUSHF   R1              ; Save -mean in the stack, all 40 bits
        LDF     *AR4++(1),R0    ; Get a random data value
        POPF    R1              ; -Mean
        POP     R1
        MPYF    R1, R0          ; random number =  -mean*ln(u)

        POPF    R1
        POP     R1
        RETS

*-----------------------------------------------------------------------
* FUNCTION DEF : _M2lnUni
* Generate a random number with exponential distribution
* Input        : None
* Register     : R0
* Output       : R0 Is the ln(uniform) random number series
*-----------------------------------------------------------------------
_M2lnUni:
        LDI     @M2SEED,R0      ; Call here for last SEED
        MPYI    @A,R0           ; Calculate r = (A*r+C)%m
        ADDI    @C,R0           ;
        AND     @MASK1, R0      ; m=2^23
        STI     R0,@M2SEED      ; Result is returned
        ADDI    1,R0            ;
        FLOAT   R0,R0           ;
        MPYF    @INVE23,R0      ; r/m
        CALL    _ln             ; ln(u)
        RETS


*-----------------------------------------------------------------------
* FUNCTION DEF : _exit
* Unusually exit
* Input        : none
* Register     : none
* Output       : none
*-----------------------------------------------------------------------
_exit:
EXITSIM B       $
```

```
        .sect "isrcode1"
*-------------------------------------------------------------------------
* FUNCTION DEF : _M1depart
* handle current departure event on M1
* Input        : None
* Register     : R0, R1, R2, R3, R4, R5, AR0, AR1, RC, RS, RE, DP
* Output       : None
*-------------------------------------------------------------------------
_M1depart:
        LDI     0,R0
        STI     R0,@COMPFIN     ;Computation not finished

        LDF     @M1SumStay,R2   ;Update part A stay in system
        ADDF    @M1CurPart+5,R2
        SUBF    @M1CurPart+3,R2
        STF     R2,@M1SumStay
        LDI     @M1SumPartA,R3  ;Part A throughput
        ADDI    1,R3
        STI     R3,@M1SumPartA
;Schedule next part departure
        LDI     @M1NumInQ ,R3   ;Is the input queue empty?
        BNZ     M1LD3           ;No, go to LD3

        LDI     0,R1
        STI     R1,@M1+1        ;Yes, set machine 1 idle, R1=0
        LDF     @BigNum+1,R3
        STF     R3,@EveList2+3  ;No next departure
        B       M1LDE
*** B   M1LDE  ;BRANCH OCCURS
M1LD3   LDI     1, R0
        STI     R0,@M1+1        ;Machine busy R0 = 1
        LDI     @M1NumInQ,R3    ;Reduce queue length
        SUBI    R0,R3
        STI     R3,@M1NumInQ
        LDF     @Time,R5        ;Update delay accumulator
        SUBF    @M1InputQ0+9,R5
        LDF     @M1SumDelay,R3
        ADDF    R5,R3
        STF     R3,@M1SumDelay

        LDI     @M1InputQ,AR0
        ADDI    6,AR0
        LDI     @M1PCurPart,AR1
        LDI     5,RC
        RPTB    M1LD6
        LDI     *AR0++,R0       ;CurPart = InputQ[1]
M1LD6   STI     R0,*AR1++

;Maintain the queue
        LDI     @M1InputQ,AR0
        ADDI    6,AR0           ;First in queue
        LDI     6,R0            ;
        ADDI3   R0,AR0,AR1      ;Next in queue
        LDI     @M1NumInQ,R0
        ADDI    1,R0            ;Fill the last position with zeros
        MPYI    6,R0            ;The total number need to be relocated
        SUBI    1,R0            ;
        LDI     R0,RC           ;Repeat (NumInQ+1)*6 times
        RPTB    M1LD7
        LDI     *AR1++,R1
M1LD7   STI     R1,*AR0++

;Schedule next departure
        LDI     2,R0
        LDI     3,R1
        CALL    _M1rerobot      ;Request robot
        ADDF    @Time,R0
        STF     R0,@M1CurPart+4
        LDF     @M1+3,R0
        CALL    _M2expon        ;Processing time
        ADDF    @M1CurPart+4,R0
```

```
        STF     R0,@EveList2+3  ;Next departure time
        STF     R0,@M1CurPart+5 ;Part finished time
        ADDF    @M1SumM1,R0     ;Update Machine busy period
        SUBF    @M1CurPart+4,R0
        STF     R0,@M1SumM1
        LDI     1,R0
        STI     R0,@COMPFIN     ;Computation finished
        LDI     @HALT,R0
        BNZ     M1LDE
        LDI     2C0h, R1
        STI     R1, @8030h      ;Start Timer 1
        LDI     1,R0
        STI     R0,@HALT
        STI     R0,*AR7
M1LDE   RETI


STACK   .word  $+1          ; Bottom of the Stack
        .end
```

Note:

1. Subroutines _ln and _INVF30 are the same as in 1Mhard1.asm
2. Only one set of programs is included here

C.1.3 Sample Data Recorded From the Prototype Simulator

Table A1 and A2 show two set of data recorded for simulating 1-workstation case. There are total 15 sets of such data generated by randomly changing seeds.

A1 Experiment No. 1 for simulating 1-workstaton (seeds 0/0)

Experiment Data for 1-Machine, 1 Part, 1 robot Simulation			
Date: Mar. 1 Experiment No.: 1			
	DESM	**PDES**	**Hardware**
Executed Cycles:	495617	441901	336031
Simulation Time:	14401	14407	14407
Seeds:	0	0	0
	0	0	0
MTBA	15	15	15
No. Part Arrivals	965	966	966
No.Part Processed	964	964	964
Avg. No in Que	0.384	0.384	0.384
Util of Mach.	0.402	0.403	0.403
Util of Robot	0.08	0.08	0.08
Avg. Stay on Mach.	12.334	12.334	12.334
Avg. Delay in Que	5.736	5.736	5.736

A2 Experiment No. 15 for simulating 1-workstaton (seeds 702749/431012)

Experiment Data for 1-Machine, 1 Part, 1 robot Simulation			
Date: Mar. 1 Experiment No.: 15			
	DESM	**PDES**	**Hardware**
Executed Cycles:	501022	452645	343201
Simulation Time:	14401	14401	14401
Seeds:	702749	702749	702749
	431012	431012	431012
MTBA	15	15	15
No. Part Arrivals	998	998	998
No.Part Processed	995	995	995
Avg. No in Que	0.363	0.363	0.363
Util of Mach.	0.415	0.415	0.415
Util of Robot	0.083	0.083	0.083
Avg. Stay on Mach.	11.832	11.832	11.832
Avg. Delay in Que	5.248	5.248	5.248

Table A3 and A4 show two set of data recorded for simulating 1-workstation case. There are total 9 sets of such data generated by changing arrival pattern.

A3 Experiment No. 1 for simulating 1-workstaton (arrival 7.5)

Experiment Data for 1-Machine, 1 Part, 1 robot Simulation			
Date: Mar. 2 Experiment No.: 1			
	DESM	**PDES**	**Hardware**
Executed Cycles:	1310800	1190962	979779
Simulation Time:	14400	14413	14413
Seeds:	0	0	0
	0	0	0
MTBA	7.5	7.5	7.5
No. Part Arrivals	1927	1927	1927
No.Part Processed	1896	1896	1896
Avg. No in Que	6.686	6.712	6.712
Util of Mach.	0.795	0.795	0.795
Util of Robot	0.158	0.158	0.158
Avg. Stay on Mach.	56.118	56.118	56.118
Avg. Delay in Que	49.584	49.584	49.584

A4 Experiment No. 9 for simulating 1-workstaton (arrival 30)

Experiment Data for 1-Machine, 1 Part, 1 robot Simulation			
Date: Mar. 2 Experiment No.: 9			
	DESM	**PDES**	**Hardware**
Executed Cycles:	289709	259978	205364
Simulation Time:	14419	14422	14422
Seeds:	0	0	0
	0	0	0
MTBA	30	30	30
No. Part Arrivals	500	500	500
No.Part Processed	498	499	499
Avg. No in Que	0.072	0.072	0.072
Util of Mach.	0.21	0.21	0.21
Util of Robot	0.042	0.042	0.042
Avg. Stay on Mach.	8.735	8.735	8.735
Avg. Delay in Que	2.075	2.075	2.075

C.2 Experiment Case for 2-workstation Product Line

C.2.1 Simulation Results from ProModel

```
-------------------------------------------------------------------------------
General Report
Output from C:\Project\FMSModel\ThesisModel\SimpleMfgLine-2M2R.MOD [DesktopFMS]
Date: Apr/13/2001   Time: 03:28:28 PM
-------------------------------------------------------------------------------
Scenario        : Normal Run
Replication     : Average
Period          : Final Report (0 sec to 240 hr Elapsed: 240 hr)
Simulation Time : 240 hr
-------------------------------------------------------------------------------
```

LOCATIONS

Location Name	Scheduled Hours	Capacity	Total Entries	Average Minutes Per Entry	Average Contents	Maximum Contents	Current Contents	% Util	
Lathe1	240	1	944.3	7.730575	0.506597	1	0.6	50.66	(Average)
Lathe1	0	0	27.3904	0.286850	0.0131165	0	0.516398	1.31	(Std. Dev.)
Mill1	240	1	1662	6.226190	0.718358	1	0.9	71.84	(Average)
Mill1	0	0	37.2559	0.132117	0.00982197	0	0.316228	0.98	(Std. Dev.)
Conveyor21	240	999999	945.4	12.576834	0.824847	4.9	1.1	16.50	(Average)
Conveyor21	0	0	26.3405	1.132412	0.0664872	0.316228	1.28668	1.33	(Std. Dev.)
Conveyor22	240	999999	720.8	22.757106	1.13963	4.6	0.8	22.79	(Average)
Conveyor22	0	0	24.8095	1.214310	0.081939	0.516398	1.0328	1.64	(Std. Dev.)
WIPQ	240	5	943.7	35.904988	2.35164	5	1.7	47.03	(Average)
WIPQ	0	0	27.7691	2.564550	0.16132	0	2.31181	3.23	(Std. Dev.)

LOCATION STATES BY PERCENTAGE (Multiple Capacity)

Location Name	Scheduled Hours	% Empty	% Partially Occupied	% Full	% Down	
Conveyor21	240	56.66	43.34	0.00	0.00	(Average)
Conveyor21	0	1.90	1.90	0.00	0.00	(Std. Dev.)
Conveyor22	240	37.75	62.25	0.00	0.00	(Average)
Conveyor22	0	3.08	3.08	0.00	0.00	(Std. Dev.)
WIPQ	240	26.75	49.72	23.53	0.00	(Average)
WIPQ	0	2.74	3.27	3.25	0.00	(Std. Dev.)

LOCATION STATES BY PERCENTAGE (Single Capacity/Tanks)

Location Name	Scheduled Hours	% Operation	% Setup	% Idle	% Waiting	% Blocked	% Down	
Lathe1	240	39.48	0.00	43.96	0.00	11.19	5.38	(Average)
Lathe1	0	1.00	0.00	1.31	0.00	1.53	0.00	(Std. Dev.)
Mill1	240	71.84	0.00	28.16	0.00	0.00	0.00	(Average)
Mill1	0	0.98	0.00	0.98	0.00	0.00	0.00	(Std. Dev.)

RESOURCES

Resource Name	Units	Scheduled Hours	Number Of Times Used	Average Minutes Per Usage	Average Minutes Travel To Use	Average Minutes Travel To Park	% Blocked In Travel	% Util	
Gryphon1	1	240	1888	0.467000	0.299779	0.000000	0.00	10.05	(Average)
Gryphon1	0	0	55.1584	0.000000	0.000077	0.000000	0.00	0.29	(Std. Dev.)
Gryphon2	1	240	3323.1	0.614959	0.214860	0.000000	0.00	19.15	(Average)
Gryphon2	0	0	74.5303	0.001693	0.001699	0.000000	0.00	0.44	(Std. Dev.)

RESOURCE STATES BY PERCENTAGE

Resource Name	Scheduled Hours	% In Use	% Travel To Use	% Travel To Park	% Idle	% Down	
Gryphon1	240	6.12	3.93	0.00	89.95	0.00	(Average)
Gryphon1	0	0.18	0.11	0.00	0.29	0.00	(Std. Dev.)
Gryphon2	240	14.19	4.96	0.00	80.85	0.00	(Average)
Gryphon2	0	0.32	0.12	0.00	0.44	0.00	(Std. Dev.)

ENTITY ACTIVITY

Entity Name	Total Exits	Current Quantity In System	Average Minutes In System	Average Minutes In Move Logic	Average Minutes Wait For Res, etc.	Average Minutes In Operation	Average Minutes Blocked	
PartA	941.4	4.1	67.716710	3.843407	24.798730	20.297050	18.777522	(Average)
PartA	28.3165	3.57305	4.003965	0.013817	2.220341	1.020468	0.998333	(Std. Dev.)
PartB	719.7	1.2	29.805335	2.626330	0.000000	14.262602	12.916403	(Average)
PartB	24.5042	1.39841	1.393976	0.009797	0.000000	1.160772	0.452342	(Std. Dev.)

ENTITY STATES BY PERCENTAGE

Entity Name	% In Move Logic	% Wait For Res, etc.	% In Operation	% Blocked	
PartA	5.69	36.57	29.99	27.74	(Average)
PartA	0.33	1.35	0.67	0.64	(Std. Dev.)
PartB	8.83	0.00	47.79	43.38	(Average)
PartB	0.42	0.00	1.90	1.58	(Std. Dev.)

C.2.2 Assembly Programs

C.2.2.1 Code running on DSP 1

```
*-----------------------------------------------------------------------
* 2MHard1.ASM -- Hardware-based simulation
* of a scenario: two machine, two part type, two robot, and considering machine breakdown
* This is the program running on Microprocessor 1, using to simulating workstation 1
* Programmed by: Dong Xu
* Date         : Jan.21, 2001
* Modified     : Mar.6, 2001
*-----------------------------------------------------------------------
        .start "data", 0x809900    ;Data section, 809900-809AFF
        .start "code", 0x809B00     ;Code section, 809B00-809EFF
        .start "isrcode", 0x809D00  ;Interrupt service routine section
        .start "shared", 0xA00000   ;Shared memory section, A00000-A003FF
        .start "shared2", 0xC00000   ;Shared memory section, C00000-C003FF
        .start "M1data", 0xB00000   ;M1 random data section, B00000-B03FFF
        .start "M2data", 0xB04000   ;M2 random data section, B04000-B07FFF

        .sect "M1data"
;M1 random number
RandNum1    .float 0.0  ;first of generated random number

        .sect "M2data"
;M2 random number
RandNum2    .float 0.0  ;first of generated random number

        .sect "data"
;Constants for Timer 0 control
EINT    .word 00002000h    ;Enable global interrupt
ETINT0  .word 00000100h    ;Enable Timer0 interrupt
PERIOD  .int 50000         ;Timer period
BISR    .word 60809D00h    ;Branch to interrupt service routine, 60 jump to 0x809DD0h
COUNT   .int 0             ;Count of Timer interrupt
TOTCYC  .int 0             ;Total cycles
SysExit .int 5             ;Exit state of the simulation,
;                            0 success, 1 event list empty, 2 Lathe 1 input queue overflow
;                            3 Mill 1 input queue overflow, 5 other fatal error, 6 ln negative
;These global variables are used to pass parameters between functions to execute the simulation
;Average of simulation results
AveInQ      .float 0.0  ;Average of number parts in queue
AveM1       .float 0.0  ;Average of machine 1 utilization
AveBlock    .float 0.0  ;Average block time of parts
AveR        .float 0.0  ;Average of robot utilization
AveStay     .float 0.0  ;Average time stay in system
AveDelay    .float 0.0  ;Average delay time of parts

NumInQ      .int 0      ;Number of parts waiting in M1 input queue
TotNum      .int 1      ;Total number of part entering
SumPartA    .int 0      ;Total finished Part A
M1PState    .int 0      ;Previous state of machine 1 before broken

SumInQ      .float 0.0  ;Sum of number parts in queue
SumM1       .float 0.0  ;Sum of machine 1 utilization
SumDelay    .float 0.0  ;Total delay time of parts
SumBlock    .float 0.0  ;Total block time of parts
SumStay     .float 0.0  ;Total time stay in system
EndTime     .float 14400.0;End simulation time, 240 hours
MeArriA     .float 15.0 ;Mean time between arrival of Part A

Time        .float 0.0  ;Current simulating time, in minutes
LastTime    .float 0.0  ;Time of last event
NxtEvet     .int 0      ;Next event type, 0=empty, 1=arrival, 2=departure, 3=machine broken
;Event list
;current event
EveList0    .int 0      ;Event type = 0(current), 1(Arrival), 2(Departure), 3(Machine Broken)
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 1      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
;arrival event
```

```
EveList1    .int 1      ;Event type = 0(current), 1(Arrival), 2(Departure), 3(Machine Broken)
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 1      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
;departure event
EveList2    .int 2      ;Event type = 0(`current), 1(Arrival), 2(Departure), 3(Machine Broken)
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 1      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
;Workstation 1 breakdown event
EveList3    .int 3      ;Event type = 0(current), 1(Arrival), 2(Departure), 3(Machine Broken)
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 1      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time

InputQ0     .space 132  ;Input queue for the workstation 1, =6*21+6 for tail

;Robot path network, using real travel time if necessary
RNetwork0   .float 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
            .float 0.0, 0.0, 0.3, 0.6, 0.9, 1.2, 1.5
            .float 0.0, 0.3, 0.0, 0.3, 0.6, 0.9, 1.2
            .float 0.0, 0.6, 0.3, 0.0, 0.3, 0.6, 0.9
            .float 0.0, 0.9, 0.6, 0.3, 0.0, 0.3, 0.6
            .float 0.0, 1.2, 0.9, 0.6, 0.3, 0.0, 0.3
            .float 0.0, 1.5, 1.2, 0.9, 0.6, 0.3, 0.0

;Define the pointers to the arrays
EveList     .word EveList0
InputQ      .word InputQ0
RNetwork    .word RNetwork0
PCurPart    .word CurPart

;Defined the structure used in the program
;Describe workstation 1
M1          .int 1      ;Machine type, 1 = lathe, 2 = mill
            .int 0      ;Machine state = 0(idle), 1(busy), 2(blocked), 3(broken)
            .int 0      ;Current operating part type, 0 = Part A, 1 = Part B
            .float 6.0  ;Processing time for Part A
            .float 0.0  ;Processing time for Part B, not available
            .float 450  ;Mean time between failure
            .float 25   ;Mean time to repair

;Current part scheduled to be processed on machine
CurPart     .int 0      ;Part type = 0(Part A), 1(Part B)
            .int 1      ;Operation required on workstation 1, 0(No), 1(Yes)
            .int 1      ;Operation required on workstation 2, 0(No), 1(Yes)
            .float 0.0  ;Time when entering the FMS
            .float 0.0  ;Processing start time to one workstation
            .float 0.0  ;Finish time of one operation on one workstation

;Describe the robot
Grephon     .int 0      ;Robot state = 0(idle), 1(occupied)
            .int 1      ;current position of robot
            .float 0.0  ;Next available time of the robot
            .float 0.0  ;Area of robot utilization

*-----------------------------------------------------------------------
*Define references
*-----------------------------------------------------------------------
M1Counter   .int 0              ;Counter of M1 random number generation
M2Counter   .int 0              ;Counter of M2 random number generation
RandTotNum  .word 0x00003FFE    ;Length limit of the data
BigRandNum  .float -15          ;Random number is unusually small

PRandNum1   .word RandNum1
PRandNum2   .word RandNum2

*-----------------------------------------------------------------------
* DEFINE CONSTANTS
*-----------------------------------------------------------------------
QLimit      .int 20             ;Length limit of the queue
```

```
WIPLimit    .int 5              ;Limit of WIP queue
NumEvet     .int 3              ;Number of event
NumPType    .int 2              ;Number of part type


;Constant used by random number generation subroutine
A       .int    1078373         ; Constants needed for RAND
C       .int    2311527         ;
M1SEED  .int    0          ;
MASK    .word   0xFF7FFFFF      ;Mask for fast inverse float
MASK1   .word   0x007FFFFF      ;Mask for exponential distribution, %2^23
INVE23  .word   0xE9000000      ;1/2^23
BigNum  .float  1.0e29          ;Constant big value
        .float  1.0e30          ;

;Internal constants for ln(u)
;Scaling coefficients for ln(1+x)
LNRM    .float  0.6931471806    ; LN(2)
C0      .float  1.0000000000    ; C0 (1.0)
;       Polynomial coefficients for ln(1+X), 0 <= X < 1.
        .float  0.9999964239    ; TOP OF C1
        .float  -0.4998741238   ; TOP OF C2
        .float  0.3317990258    ; TOP OF C3
        .float  -0.2407338084   ; TOP OF C4
        .float  0.1676540711    ; TOP OF C5
        .float  -0.0953293897   ; TOP OF C6
        .float  0.0360884937    ; TOP OF C7
C8      .float  -0.0064535442   ; TOP OF C8
AC8     .word   C8

*-------------------------------------------------------------------------------
*Special signals for synchronization purpose, should be stored in the shared memory,
*accessed by to both Microprocessor
*-------------------------------------------------------------------------------
    .sect "shared"
M2Finish    .int 1      ;Indicate M2 finish the previous part departure event
M2QFull     .int 0      ;Signal of WIP queue, 0 = not full, 1 = full
PartArr     .int 0      ;Indicate Part A leave from Machine 1 to 2
SafeTime    .float 0.0  ;Indicate how far simulation clock on M2 can advance
M2D_Time    .float 0.0  ;Scheduled next part departure time on machine 2

;Departure message from Machine 1 to Machine
M1_M2Dev    .int 0      ;Event type = 0(current), 1(Arrival), 2(Departure), 3(Machine Broken)
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 1      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
;Current Part transported from Machine 1 to Machine 2
M1_M2Par    .int 0      ;Part type = 0(Part A), 1(Part B)
            .int 1      ;Operation required on workstation 1, 0(No), 1(Yes)
            .int 1      ;Operation required on workstation 2, 0(No), 1(Yes)
            .float 0.0  ;Time when entering the FMS
            .float 0.0  ;Arrival time to one workstation
            .float 0.0  ;Finish time of one operation on one workstation
SharedMem   .word M2Finish
Barrier1    .int 0

    .sect "shared2"
Barrier2    .int 0      ;Barrier counter to start the process
Finished    .int 0      ;Simulation Execution finished
SharedMem2  .word Barrier2

    .sect "code"
    .entry  _main
*-------------------------------------------------------------------------------
* FUNCTION DEF : _main
* Control the advance of the program
*-------------------------------------------------------------------------------
_main:
        LDP     @_main          ;Set Data Segment Pointer
        LDI     @STACK,SP       ;Set stack pointer
        CALL    _M1RandNum      ;Prepare random number
```

```
        LDI     @PRandNum1,AR3  ;Index of random data series
        CALL    _initial        ;Initialize the simulation, especially data structure
        CALL    _barrier
LM1     LDF     @Time,R0
        CMPF    @EndTime,R0     ;Does simulation end?
        BGT     LM2

        CALL    _timing         ;Determine the next event type

;Invoke the appropriate event handle function
        LDI     @PSUB,AR0
        ADDI    @NxtEvet,AR0
        LDI     *AR0,R0
        CALLU   R0              ;Handle part arrival, departure, or machine breakdown events
        B       LM1
LM2
        CALL    _report         ;Report simulation results

        LDI     0,R0
        STI     R0,@SysExit     ;Successfully run
        LDP     SharedMem2
        LDI     1,R0
        STI     R0,@Finished    ;Execution finished
END     B       $               ;End of the program

*-------------------------------------------------------------------------
* FUNCTION DEF : _M1RandNum
* Fill the robot path network, initialize the data structures and the first arrival
* Input        : None
* Register     : R0,R1,AR5
* Output       : None
*-------------------------------------------------------------------------
_M1RandNum:
        LDI     @PRandNum1, AR5 ;AR5 is the index for storage of M1 random numbers
        LDI     0,R0
        STI     R0,@M1Counter   ;Initialize counter

M1LR1   LDI     @M1Counter,R0
        CMPI    @RandTotNum,R0  ;Does process end?
        BGE     M1LR4

;Increase counter
        LDI     @M1Counter,R0
        ADDI    1,R0
        STI     R0,@M1Counter
;Generate random number for M1
        CALL    _M1lnUni         ;
        STF     R0, *AR5++(1)    ;Store the random number
        LDF     *-AR5(1),R2      ;Read the stored random number
        PUSHF   R0
        POPF    R0               ;Set R0 to the 32-bit
        CMPF    R2,R0            ;
        BZ      M1LR1              ;If same, continue
        LDI     11, R1
        STI     R1, @SysExit     ;Indicate M1 read/write wrong
        CALL    _exit

M1LR4   LDF     0.0,R0
        STF     R0,*AR5          ;End of data
        LDI     @M1Counter,R0
        STI     R0,*+AR5(1)      ;Total number of data
        LDI     @PRandNum1,AR5
        LDI     0,R1             ;Index
        STI     R1,@M1Counter
M1LR5   LDF     *AR5++(1),R0,
        BP      M1LR6              ;Positive?
        CMPF    @BigRandNum,R0
        BLT     M1LR7              ;Too small?
        LDI     @M1Counter,R1
        ADDI    1,R1
```

```
        STI     R1,@M1Counter
        CMPI    @RandTotNum,R1
        BLT     M1LR5

        LDI     10,R0
        STI     R0,@SysExit      ;Successfully run
        RETS
M1LR6   LDI     14,R0
        STI     R0,@SysExit      ;positive number found
        CALL    _exit
M1LR7   LDI     15,R0
        STI     R0,@SysExit      ;small number found
        CALL    _exit


*-------------------------------------------------------------------------
* FUNCTION DEF : _initial
* Fill the robot path network, initialize the data structures and the first arrival
* Input        : None
* Register     : R1, AR0, RC, RS, RE
* Output       : None
*-------------------------------------------------------------------------
_initial:
;Describe the current part on workstation 1, InputQ[0] is the storage of next part
        LDI     0,R1
        STI     R1,@InputQ0     ;Part A
        LDI     1,R1
        STI     R1,@InputQ0+1   ;Part A need turning operation on workstation 1
        STI     R1,@InputQ0+2   ;Part A need drilling operation on workstation 2
        LDF     @MeArriA,R0
        CALL    _expon
        STF     R0,@InputQ0+3  ;Part A first arrival time
        LDF     0.0,R1
        STF     R1,@InputQ0+4   ;Part A start on workstation 1, depending on robot
        STF     R1,@InputQ0+5   ;Time to finish operation on workstation 1 does not known yet

        LDF     @InputQ0+3,R1
        STF     R1,@EveList1+3  ;Schedule the arrive time
        LDF     @BigNum+1,R1
        STF     R1,@EveList2+3  ;Schedule the initial departure time to be infinite
        LDF     @M1+5,R1
        STF     R1,@EveList3+3  ;Schedule first machine down
        RETS

*-------------------------------------------------------------------------
* FUNCTION DEF : _timing
* schedule the next event, synchronize and advance the simulation clock
* Input        : None
* Register     : R0, R1, R2, AR0, DP
* Output       : None
*-------------------------------------------------------------------------
_timing:
        LDF     @BigNum,R1      ;A initial min time
        LDI     0,R0
        STI     R0,@NxtEvet    ;Next event type = 0
;Determine the minimum time among event list and set the next event type
;If two event happen at the same time, handle machine broken event first, then
;departure event, and the last is arrival event
        LDI     1,R0            ;Iteration index
        LDI     @EveList,AR0
        ADDI    3,AR0           ;Point to the first Event List E_Time
LT1     CMPF    *++AR0(4),R1    ;Event List[i].E_Time <= min_time?
        BLT     LT2
        LDF     *AR0,R1         ;Minimum time is R1
        STI     R0,@NxtEvet     ;Next Event No.
LT2     ADDI    1,R0
        CMPI    3,R0            ;Need to compare all 3 events
        BLE     LT1

        LDI     @NxtEvet,R0     ;If event list is empty
        BNZ     LT3
```

```
        LDI     1,R0            ;Exit(1), event list is empty
        STI     R0,@SysExit
        CALL    _exit           ;If even list is empty, stop the simulation and exit
LT3     CMPI    2,R0            ;If next event is a departure, then check if previous one has
                                ;been processed by microprocessor 2 or not
        BNZ     LT5
        LDP     SharedMem       ;Point to the shared memory
LT4     LDI     @M2Finish,R0    ;Check if previous departure has been processed or not?
        CMPI    1,R0
        BNZ     LT4             ;If not, wait until microprocessor 2 processed previous departure
        LDP     _main           ;Restore the data segment pointer
LT5     STF     R1,@Time        ;Update Simulation clock
        LDF     @LastTime,R0    ;Last event time
        SUBF3   R0,R1,R2        ;Compute time since last event
        STF     R1, @LastTime   ;Save last event time
        LDP     SharedMem       ;Point to the shared memory
        STF     R1,@SafeTime    ;Safe time is current time
                                ;Microprocessor 2 can process events before this safe time
        LDP     _main           ;Restore to the data segment
        FLOAT   @NumInQ,R0
        MPYF    R2,R0
        ADDF    @SumInQ,R0
        STF     R0,@SumInQ      ;Update area of number in queue
        RETS

*--------------------------------------------------------------------------
* FUNCTION DEF : _arrive
* schedule the next arrival event and handle current arrival event
* Input        : None
* Register     : R0, R1, R2, AR0, AR1
* Output       : None
*--------------------------------------------------------------------------
_arrive:
        LDI     @M1+1,R0        ;Check to see if workstation 1 is idle?
        BNZ     LA1             ;If not idle, go to LA1
        LDI     1,R0            ;If idle
        STI     R0,@M1+1        ;Set Machine 1 busy
        LDI     2,R1            ;Request robot, Load part from queue to machine 1
        CALL    _rerobot        ;R0 = initial position, R1 = destination position
                                ;Travel time return in R0
        LDF     @Time,R1
        STF     R1,@CurPart+3   ;Current part entering time
        ADDF3   R0,R1,R2
        STF     R2,@CurPart+4   ;Current process starting time
        LDF     @M1+3,R0
        CALL    _expon          ;Processing time for the part
        ADDF    @CurPart+4,R0
        STF     R0,@EveList2+3  ;Schedule a Part 1 departure after processing
                                ;considering the travel time of robot
        STF     R0,@CurPart+5   ;Current part finishing time
        ADDF    @SumM1,R0
        SUBF    @CurPart+4,R0
        STF     R0,@SumM1       ;Update Machine busy period
        B       LA4
*** B   LA4 ;BRANCH OCCURS
;If machine is not available (either busy, blocked or broken), put the current part in queue
LA1     LDI     @NumInQ,R1
        ADDI    1,R1
        STI     R1,@NumInQ      ;Increase the length of queue
        CMPI    20,R1
        BLE     LA2
        LDI     2,R0            ;Queue is overflow, exit 2
        STI     R0,@SysExit
        CALL    _exit
LA2     LDI     @NumInQ,AR0     ;InputQ[NumInQ] = InputQ[0]
        MPYI    6,AR0
        ADDI    @InputQ,AR0
        LDI     @InputQ,AR1
        LDI     5,RC
        RPTB    LA3
        LDI     *AR1++,R0       ;
```

```
LA3     STI     R0,*AR0++
;Schedule next Part A arrival
LA4     LDF     @MeArriA,R0
        CALL    _expon
        ADDF    @Time,R0
        STF     R0,@InputQ0+3
        STF     R0,@EveList1+3  ;Next arrival time
        LDI     @TotNum,R1      ;Update total entering part number
        ADDI    1,R1
        STI     R1,@TotNum
        RETS

*--------------------------------------------------------------------------
* FUNCTION DEF : _depart
* handle current departure event
* Input        : None
* Register     : R0, R1, R2, R3, R4, R5, AR0, AR1, RC, RS, RE, DP
* Output       : None
*--------------------------------------------------------------------------
_depart:
        LDI     @M1+1,R1        ;
        CMPI    3,R1            ;Is machine 1 broken?
        BNZ     LD1
        LDF     @EveList3+3,R1
        STF     R1,@CurPart+5   ;Update the finished time for current part
        STF     R1,@EveList2+3  ;Postponed current departure until the machine is up
        B       LDE
*** B   LD5  ;BRANCH OCCURS
LD1     LDP     SharedMem       ;Point to the shared memory
        LDI     @M2QFull,R0
        CMPI    1,R0            ;Check to see if Machine 2 WIP queue is full
        BNZ     LD2
        LDF     @M2D_Time,R1
        LDP     _main           ;Restore the data segment pointer
        CMPF    @EveList2+3,R1  ;If Departure time < M2D_Time?
        BLE     LDE             ;No, wait until M2 process the next departure
        LDF     R1,R5           ;Yes, save the possible departure time
        SUBF    @Time,R5        ;Blocked time period
        LDF     @SumBlock,R0
        ADDF    R5,R0
        STF     R0,@SumBlock    ;Update the total block accumulator
        STF     R1,@CurPart+5   ;Update the finished time for current part
;Reschedule the current departure to the new possible departure time
        STF     R1,@EveList2+3
        LDI     2,R1            ;Machine 2 WIP queue full, M1 blocked
        STI     R1,@M1+1
        B       LDE
*** B   LD5  ;BRANCH OCCURS
;Machine is up. Schedule a Part A leave from M1 to M2.
;Wait until microprocessor 2 read the previous messages
LD2     LDP     SharedMem       ;Point to the shared memory
        LDI     @PartArr,R0
        CMPI    1,R0            ;Wait until microprocessor 2 read previous message
        BZ      LD2
        LDP     _main           ;Restore the memory segment pointer
        LDI     @EveList2,R0    ;M1_M2Dev = EveList[2]
        LDI     @EveList2+1,R1
        LDI     @EveList2+2,R2
        LDF     @EveList2+3,R3
        LDP     SharedMem       ;Point to the shared memory
        STI     R0,@M1_M2Dev
        STI     R1,@M1_M2Dev+1
        STI     R2,@M1_M2Dev+2
        STF     R3,@M1_M2Dev+3
        LDP     _main           ;Point to the memory segment
        LDI     @CurPart,R0     ;M1_M2Par = CurPart
        LDI     @CurPart+1,R1
        LDI     @CurPart+2,R2
        LDF     @CurPart+3,R3
        LDF     @CurPart+4,R4
        LDF     @CurPart+5,R5
```

```
        LDP     SharedMem       ;Point to the shared memory
        STI     R0,@M1_M2Par
        STI     R1,@M1_M2Par+1
        STI     R2,@M1_M2Par+2
        STF     R5,@M1_M2Par+3
        STF     R4,@M1_M2Par+4
        STF     R5,@M1_M2Par+5
        LDI     1,R0
        STI     R0,@PartArr     ;Inform machine 2 part A arrival
        LDI     0,R1
        STI     R1,@M2Finish    ;Event not processed by microprocessor 2
        LDP     _main           ;Restore data segment pointer

        LDF     @SumStay,R2     ;Update part A stay in system
        ADDF    @CurPart+5,R2
        SUBF    @CurPart+3,R2
        STF     R2,@SumStay
        LDI     @SumPartA,R3    ;Part A throughput
        ADDI    1,R3
        STI     R3,@SumPartA
;Schedule next part departure
        LDI     @NumInQ ,R3     ;Is the input queue empty?
        BNZ     LD3             ;No, go to LD3
        STI     R1,@M1+1        ;Yes, set machine 1 idle, R1=0
        LDF     @BigNum+1,R3
        STF     R3,@EveList2+3  ;No next departure
        B       LDE
*** B   LD5  ;BRANCH OCCURS
LD3     LDI     1, R0
        STI     R0,@M1+1        ;Machine busy R0 = 1
        LDI     @NumInQ,R3      ;Reduce queue length
        SUBI    R0,R3
        STI     R3,@NumInQ
        LDF     @Time,R5        ;Update delay accumulator
        SUBF    @InputQ0+9,R5
        LDF     @SumDelay,R3
        ADDF    R5,R3
        STF     R3,@SumDelay

        LDI     @InputQ,AR0
        ADDI    6,AR0
        LDI     @PCurPart,AR1
        LDI     5,RC
        RPTB    LD6
        LDI     *AR0++,R0       ;CurPart = InputQ[1]
LD6     STI     R0,*AR1++

;Maintain the queue
        LDI     @InputQ,AR0
        ADDI    6,AR0           ;First in queue
        LDI     6,R0            ;
        ADDI3   R0,AR0,AR1      ;Next in queue
        LDI     @NumInQ,R0
        ADDI    1,R0            ;Fill the last position with zeros
        MPYI    6,R0            ;The total number need to be relocated
        SUBI    1,R0            ;
        LDI     R0,RC           ;Repeat (NumInQ+1)*6 times
        RPTB    LD7
        LDI     *AR1++,R1
LD7     STI     R1,*AR0++

;Schedule next departure
        LDI     2,R0
        LDI     3,R1
        CALL    _rerobot        ;Request robot
        ADDF    @Time,R0
        STF     R0,@CurPart+4
        LDF     @M1+3,R0
        CALL    _expon          ;Processing time
        ADDF    @CurPart+4,R0
        STF     R0,@EveList2+3  ;Next departure time
```

```
        STF     R0,@CurPart+5   ;Part finished time
        ADDF    @SumM1,R0       ;Update Machine busy period
        SUBF    @CurPart+4,R0
        STF     R0,@SumM1
LDE     RETS

*---------------------------------------------------------------------------
* FUNCTION DEF : _down
* handle machine break down and repair
* Input        : None
* Register     : R0, R1, R2, AR0, AR1
* Output       : None
*---------------------------------------------------------------------------
_down:
        LDI     @M1+1,R0
        CMPI    3,R0            ;Is machine current broken?
        BNZ     LO1             ;
        LDI     @M1PState,R0    ;Machine up, restore machine previous state
        STI     R0,@M1+1
        LDF     @Time,R1        ;Working period
        ADDF    @M1+5,R1
        STF     R1,@EveList3+3  ;Schedule next break down time
        B       LOE
LO1     STI     R0,@M1PState    ;Save current machine status
        LDI     3,R1
        STI     R1,@M1+1        ;Set machine down
        LDF     @Time,R2        ;Repair period
        ADDF    @M1+6,R2
        STF     R2,@EveList3+3  ;Schedule next machine up
LOE     RETS

*---------------------------------------------------------------------------
* FUNCTION DEF : _rerobot
* handle robot request, return robot travel time.
* Input        : R0,initial position, R1, destination position
* Register     : R0, R1, AR0, IR0, IR1, (only AR0,IR0,IR1 restored)
* Output       : R0 travel time
*---------------------------------------------------------------------------
_rerobot:
;Calculate travel time, robot need to travel from current position to initial position,
;then from initial position to the destination
        PUSH    AR0
        PUSH    IR0
        PUSH    IR1

        LDI     @Grephon+1,IR0
        MPYI    7,IR0
        ADDI    R0,IR0                  ;From current position to starting position
        LDI     @RNetwork,AR0
        LDI     R0,IR1
        MPYI    7,IR1
        ADDI    R1,IR1                  ;From starting position to destination
;Robot traveltime = RNetwork[Grephon.R_Pos][initpos] + RNetwork[initpos][destpos]
        ADDF3   *+AR0(IR1),*+AR0(IR0),R0
        STI     R1,@Grephon+1           ;Next robot position
        LDF     @Time,R1
        CMPF    @Grephon+2,R1
        BGE     LRE1
        ADDF    @Grephon+2,R0           ;Robot is occupied, considering some waiting time
        SUBF    R1,R0
;Next time robot stay at initial position and available time is current time plus travel time
LRE1    ADDF    R0,R1                   ;Next available time
        STF     R1,@Grephon+2
        LDF     2.0,R1
        MPYF    R0,R1
        ADDF    @Grephon+3,R1           ;Update robot utilization
        STF     R1,@Grephon+3

        POP     IR1
        POP     IR0
        POP     AR0
```

```
        RETS

*-----------------------------------------------------------------------
* FUNCTION DEF : _report
* report simulation results
* Input        : None
* Register     : R0, R2, R5
* Output       : None
*-----------------------------------------------------------------------
_report:
        LDF     @EndTime,R0
        CALL    INV_F30
        RND     R0,R5           ;R5=1/Time

        LDF     @SumM1,R0
        MPYF    R5,R0
        STF     R0,@AveM1       ;Utilization of M1

        LDF     @Grephon+3,R0
        MPYF    R5,R0
        STF     R0,@AveR        ;Utilization of robot

        LDF     @SumBlock,R0
        MPYF    R5,R0
        STF     R0,@AveBlock    ;Average blocked

        LDF     @SumInQ,R0
        MPYF    R5,R0
        STF     R0,@AveInQ      ;Average waiting in queue

        FLOAT   @SumPartA,R0
        CALL    INV_F30
        RND     R0,R2           ;R2 = 1/SumPart
        LDF     @SumDelay,R0
        MPYF    R2,R0
        STF     R0,@AveDelay    ;Average delay

        LDF     @SumStay,R0
        MPYF    R2,R0
        STF     R0,@AveStay     ;Average stay in FMS
        RETS

*-----------------------------------------------------------------------
* FUNCTION DEF : _exit
* Unusually exit
* Input        : none
* Register     : none
* Output       : none
*-----------------------------------------------------------------------
_exit:
EXITSIM B       $

*-----------------------------------------------------------------------
* FUNCTION DEF : _barrier
* Unusually exit
* Input        : none
* Register     : R0
* Output       : none
*-----------------------------------------------------------------------
_barrier:
        LDP     SharedMem
        LDI     @Barrier1,R0
        ADDI    1,R0
        STI     R0,@Barrier1
B1      LDI     @Barrier1,R0
        CMPI    2,R0
        BLT     B1
        LDP     SharedMem2
        LDI     @Barrier2,R0
        ADDI    1,R0
```

```
        STI     R0,@Barrier2
B2      LDI     @Barrier2,R0
        CMPI    2,R0
        BLT     B2
        LDP     @_main
        RETS

*-----------------------------------------------------------------------
* FUNCTION DEF : _expon
* Generate a random number with exponential distribution
* Input        : R0 is the mean
* Register     : R1
* Output       : R0 is the random number with exponential distribution
*-----------------------------------------------------------------------
_expon:
*Fast 32 bit uniform random number generator
        PUSH    R1
        PUSHF   R1

        NEGF    R0, R1          ; R1 is the negative of the mean
        PUSH    R1
        PUSHF   R1              ; Save -mean in the stack, all 40 bits
        LDF     *AR3++(1),R0    ; Get a random number value
        POPF    R1              ; -Mean
        POP     R1
        MPYF    R1, R0          ; random number =  -mean*ln(u)

        POPF    R1
        POP     R1
        RETS

*-----------------------------------------------------------------------
* FUNCTION DEF : _M1lnUni
* Generate a random number with exponential distribution
* Input        : None
* Register     : R0
* Output       : R0 Is the ln(uniform) random number series
*-----------------------------------------------------------------------
_M1lnUni:
        LDI     @M1SEED,R0      ; Call here for last SEED
        MPYI    @A,R0           ; Calculate r = (A*r+C)%m
        ADDI    @C,R0           ;
        AND     @MASK1, R0      ; m=2^23
        STI     R0,@M1SEED      ; Result is returned
        ADDI    1,R0            ;
        FLOAT   R0,R0           ;
        MPYF    @INVE23,R0      ; r/m
        CALL    _ln             ; ln(u)
        RETS



*-----------------------------------------------------------------------------
*Define references
*-----------------------------------------------------------------------------
*Define the entry point of subroutines
SUB     .word _main
        .word _arrive
        .word _depart
        .word _down
PSUB    .word SUB

        .sect "isrcode"
*Interrupt Service Routine
_ISRCount:
        PUSH DP
        PUSH R0
        PUSHF R0
        LDP _main
        LDI @COUNT, R0
```

```
        ADDI 1, R0
        STI R0, @COUNT
        POPF R0
        POP R0
        POP DP
        RETI

STACK   .word  $+1          ; Bottom of the Stack
        .end
```

C.2.2.2 Code running on DSP 2

```
*-------------------------------------------------------------------
* 2MHard2.ASM -- Hardware-based Simulation
* of a scenario: two machine, two part type, two robot, and considering machine breakdown
* This is the program running on Microprocessor 2, using to simulating workstation 2
* Programmed by: Dong Xu
* Date         : Jan. 21, 2001
* Modified     : Mar. 7, 2001
*-------------------------------------------------------------------
        .start "data", 0x809900    ;Data section, 809900-809AFF
        .start "code", 0x809B00     ;Code section, 809B00-809EFF
        .start "shared", 0xA00000   ;Shared memory section, A00000-A003FF
        .start ".text", 0xB00000    ;Main text section, B00000-B007FFF
        .start "M1data", 0xB00000   ;M1 random data section, B00000-B03FFF
        .start "M2data", 0xB04000   ;M2 random data section, B04000-B07FFF

        .sect "M1data"
;M1 random number
RandNum1    .float 0.0  ;first of generated random number

        .sect "M2data"
;M2 random number
RandNum2    .float 0.0  ;first of generated random number

        .sect "data"
;These global variables are used to pass parameters between functions to execute the simulation
;Average of simulation results
SysExit     .int 5      ;Exit state of the simulation
;                            0 success, 1 event list empty, 2 Lathe 1 input queue overflow
;                            3 Mill 1 input queue overflow, 5 other fatal error, 6 ln negative
AveInQ      .float 0.0  ;Average of number parts in queue for part A
            .float 0.0  ;Part B
AveM2       .float 0.0  ;Average of machine 2 utilization
AveR        .float 0.0  ;Average of robot utilization
AveStay     .float 0.0  ;Average time stay in system
            .float 0.0
AveDelay    .float 0.0  ;Average delay time of parts
            .float 0.0

NumInQ      .int 0      ;Number of parts waiting in M2 input queue
            .int 0
TotNum      .int 0      ;Total number of parts entering
            .int 1
SumPart     .int 0      ;Total finished Parts
            .int 0

SumInQ      .float 0.0  ;Sum of number parts in queues
            .float 0.0
SumM2       .float 0.0  ;Sum of machine 2 utilization
SumDelay    .float 0.0  ;Total delay time of parts
            .float 0.0
SumStay     .float 0.0  ;Total time stay in system
            .float 0.0
EndTime     .float 14400.0;End simulation time, 240 hours
MeArriB     .float 20.0 ;Mean time between arrival of Part B

Time        .float 0.0  ;Current simulating time, in minutes
LastTime    .float 0.0  ;Time of last event
NxtEvet     .int 0      ;Next event type, 0=empty, 1=part B arrival, 2=part A arrival
                        ;3=parts departure
;Event list
;current event
EveList0    .int 0      ;Event type = 0(current), 1(Arrival1), 2(Arrival2), 3(Departure)
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 2      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
;Part B arrival event
EveList1    .int 1      ;Event type = 0(current), 1(Arrival1), 2(Arrival2), 3(Departure)
            .int 1      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 2      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
```

```
;Part A arrival event
EveList2    .int 2      ;Event type = 0(current), 1(Arrival1), 2(Arrival2), 3(Departure))
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 2      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
;departure event
EveList3    .int 3      ;Event type = 0(current), 1(Arrival1), 2(Arrival2), 3(Departure)
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 2      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time

WIPQ0       .space 42   ;Working-in-process queue for part A, = 6*6+6 = 42
InputQ0     .space 132  ;Input queue for part B, =6*21+6 for tail

;Robot path network, using real travel time if necessary
RNetwork0   .float 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
            .float 0.0, 0.0, 0.3, 0.6, 0.9, 1.2, 1.5
            .float 0.0, 0.3, 0.0, 0.3, 0.6, 0.9, 1.2
            .float 0.0, 0.6, 0.3, 0.0, 0.3, 0.6, 0.9
            .float 0.0, 0.9, 0.6, 0.3, 0.0, 0.3, 0.6
            .float 0.0, 1.2, 0.9, 0.6, 0.3, 0.0, 0.3
            .float 0.0, 1.5, 1.2, 0.9, 0.6, 0.3, 0.0

;Define the pointers to the arrays
PAveInQ     .word AveInQ
PAveDelay   .word AveDelay
PAveStay    .word AveStay
PNumInQ     .word NumInQ
PTotNum     .word TotNum
PSumInQ     .word SumInQ
PSumPart    .word SumPart
PSumDelay   .word SumDelay
PSumStay    .word SumStay
PM2         .word M2
PCurPart    .word CurPart
EveList     .word EveList0
InputQ      .word InputQ0
WIPQ        .word WIPQ0
RNetwork    .word RNetwork0

;Defined the structure used in the program
;Describe workstation 2
M2          .int 2      ;Machine type, 1 = lathe, 2 = mill
            .int 0      ;Machine state = 0(idle), 1(busy), 2(blocked), 3(broken)
            .int 0      ;Current operating part type, 0 = Part A, 1 = Part B
            .float 8.0  ;Processing time for Part A
            .float 4.0  ;Processing time for Part B, not available
            .float 0.0  ;Mean time between failure
            .float 0.0  ;Mean time to repair

;Current part scheduled to be processed on machine
CurPart     .int 1      ;Part type = 0(Part A), 1(Part B)
            .int 0      ;Operation required on workstation 1, 0(No), 1(Yes)
            .int 1      ;Operation required on workstation 2, 0(No), 1(Yes)
            .float 0.0  ;Time when entering the FMS
            .float 0.0  ;Processing start time to one workstation
            .float 0.0  ;Finish time of one operation on one workstation

;Describe the robot
Grephon     .int 0      ;Robot state = 0(idle), 1(occupied)
            .int 4      ;current position of robot
            .float 0.0  ;Next available time of the robot
            .float 0.0  ;Area of robot utilization

*-----------------------------------------------------------------------
*Define references
*-----------------------------------------------------------------------
M1Counter   .int 0              ;Counter of M1 random number generation
M2Counter   .int 0              ;Counter of M2 random number generation
RandTotNum  .word 0x00003FFE    ;Length limit of the data
BigRandNum  .float -15          ;Random number is unusually small
```

```
PRandNum1   .word RandNum1
PRandNum2   .word RandNum2

*-------------------------------------------------------------------------
* DEFINE CONSTANTS
*-------------------------------------------------------------------------
QLimit      .int 20             ;Length limit of the queue
WIPLimit    .int 5              ;Limit of WIP queue
NumEvet     .int 3              ;Number of event
NumPType    .int 2              ;Number of part type

;Constant used by random number generation subroutine
A       .int    1078373         ; Constants needed for RAND
C       .int    2311527         ;
M2SEED  .int    0       ;
MASK    .word   0xFF7FFFFF      ;Mask for fast inverse float
MASK1   .word   0x007FFFFF      ;Mask for exponential distribution, %2^23
INVE23  .word   0xE9000000      ;1/2^23
BigNum  .float  1.0e29          ;Constant big value
        .float  1.0e30          ;

;Internal constants for ln(u)
;Scaling coefficients for ln(1+x)
LNRM    .float  0.6931471806    ; LN(2)
C0      .float  1.0000000000    ; C0 (1.0)
;       Polynomial coefficients for ln(1+X), 0 <= X < 1.
        .float  0.9999964239    ; TOP OF C1
        .float  -0.4998741238   ; TOP OF C2
        .float  0.3317990258    ; TOP OF C3
        .float  -0.2407338084   ; TOP OF C4
        .float  0.1676540711    ; TOP OF C5
        .float  -0.0953293897   ; TOP OF C6
        .float  0.0360884937    ; TOP OF C7
C8      .float  -0.0064535442   ; TOP OF C8
AC8     .word   C8

*-------------------------------------------------------------------------------------
*Special signals for synchronization purpose, should be stored in the shared memory,
*accessed by to both Microprocessor
*-------------------------------------------------------------------------------------
    .sect "shared"
M2Finish    .int 1      ;Indicate M2 finish the previous part departure event
M2QFull     .int 0      ;Signal of WIP queue, 0 = not full, 1 = full
PartArr     .int 0      ;Indicate Part A leave from Machine 1 to 2
SafeTime    .float 0.0  ;Indicate how far simulation clock on M2 can advance
M2D_Time    .float 0.0  ;Scheduled next part departure time on machine 2

;Departure message from Machine 1 to Machine
M1_M2Dev    .int 0      ;Event type = 0(current), 1(Arrival), 2(Departure), 3(Machine Broken)
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 1      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
;Current Part transported from Machine 1 to Machine 2
M1_M2Par    .int 0      ;Part type = 0(Part A), 1(Part B)
            .int 1      ;Operation required on workstation 1, 0(No), 1(Yes)
            .int 1      ;Operation required on workstation 2, 0(No), 1(Yes)
            .float 0.0  ;Time when entering the FMS
            .float 0.0  ;Arrival time to one workstation
            .float 0.0  ;Finish time of one operation on one workstation
SharedMem   .word M2Finish
Barrier1    .int 0

    .sect "code"
    .entry  _main
*-------------------------------------------------------------------------------
* FUNCTION DEF : _main
* Control the advance of the program
*-------------------------------------------------------------------------------
_main:
        LDP     @_main          ;Set Data Segment Pointer
```

```
        LDI     @STACK,SP       ;Set stack pointer
        CALL    _M2RandNum      ;Prepare random number
        LDI     @PRandNum2,AR4  ;Index of random data series

        CALL    _initial        ;Initialize the simulation, especially data structure
        CALL    _barrier

LM1     LDF     @Time,R0
        CMPF    @EndTime,R0     ;Does simulation end?
        BGT     LM2

        CALL    _timing         ;Determine the next event type

;Invoke the appropriate event handle function
        LDI     @PSUB,AR0
        ADDI    @NxtEvet,AR0
        LDI     *AR0,R0
        CALLU   R0              ;Handle parts arrival, departure
        B       LM1
LM2
        CALL    _report         ;Report simulation results

        LDI     0,R0
        STI     R0,@SysExit     ;Successfully run

END     B       $               ;End of the program


*-------------------------------------------------------------------------
* FUNCTION DEF : _M2RandNum
* Fill the robot path network, initialize the data structures and the first arrival
* Input        : None
* Register     : R0,R1,AR6
* Output       : None
*-------------------------------------------------------------------------
_M2RandNum:
        LDI     @PRandNum2, AR6 ;AR5 is the index for storage of M1 random numbers
        LDI     0,R0
        STI     R0,@M2Counter   ;Initialize counter

M2LR1   LDI     @M2Counter,R0
        CMPI    @RandTotNum,R0  ;Does process end?
        BGE     M2LR4

;Increase counter
        LDI     @M2Counter,R0
        ADDI    1,R0
        STI     R0,@M2Counter
;Generate random number for M1
        CALL    _M2lnUni        ;
        STF     R0, *AR6++(1)   ;Store the random number
        LDF     *-AR6(1),R2     ;Read the stored random number
        PUSHF   R0
        POPF    R0              ;Set R0 to the 32-bit
        CMPF    R2,R0           ;
        BZ      M2LR1             ;If same, continue
        LDI     11, R1
        STI     R1, @SysExit    ;Indicate M1 read/write wrong
        CALL    _exit

M2LR4   LDF     0.0,R0
        STF     R0,*AR6         ;End of data
        LDI     @M2Counter,R0
        STI     R0,*+AR6(1)     ;Total number of data
        LDI     @PRandNum2,AR6
        LDI     0,R1            ;Index
        STI     R1,@M2Counter
M2LR5   LDF     *AR6++(1),R0,
        BP      M2LR6             ;Positive?
        CMPF    @BigRandNum,R0
        BLT     M2LR7             ;Too small?
```

```
        LDI     @M2Counter,R1
        ADDI    1,R1
        STI     R1,@M2Counter
        CMPI    @RandTotNum,R1
        BLT     M2LR5

        LDI     10,R0
        STI     R0,@SysExit      ;Successfully run
        RETS
M2LR6     LDI     14,R0
        STI     R0,@SysExit      ;positive number found
        CALL    _exit
M2LR7     LDI     15,R0
        STI     R0,@SysExit      ;small number found
        CALL    _exit


*-------------------------------------------------------------------------
* FUNCTION DEF : _initial
* Fill the robot path network, initialize the data structures and the first arrival
* Input        : None
* Register     : R1, AR0, RC, RS, RE
* Output       : None
*-------------------------------------------------------------------------
_initial:
;Describe the current part on workstation 2, WIPQ0[0] is the storage of next part A
        LDI     0,R1
        STI     R1,@WIPQ0       ;Part A
        LDI     1,R1
        STI     R1,@WIPQ0+1     ;Part A need turning operation on workstation 1
        STI     R1,@WIPQ0+2     ;Part A need drilling operation on workstation 2
        LDF     0.0,R1
        STF     R1,@WIPQ0+3     ;Part A first arrival time
        STF     R1,@WIPQ0+4     ;Part A start on workstation 2
        STF     R1,@WIPQ0+5     ;Time to finish operation on workstation 2
;Describe the part B on workstation 2, InputQ0[0] is the storage of next part
        LDI     1,R0
        STI     R0,@InputQ0     ;Part B
        LDI     0,R1
        STI     R1,@InputQ0+1   ;Part B do not need turning operation on workstation 1
        STI     R0,@InputQ0+2   ;Part B need drilling operation on workstation 2
        LDF     @MeArriB,R0
        CALL    _expon
        STF     R0,@InputQ0+3   ;Part B first arrival time
        LDF     0.0,R1
        STF     R1,@InputQ0+4   ;Part B start on workstation 2, depending on robot
        STF     R1,@InputQ0+5   ;Time to finish operation on workstation 2 does not known yet
;Schedule first arrivals
        LDF     @InputQ0+3,R1
        STF     R1,@EveList1+3  ;Schedule the part B arrive time
        LDF     @BigNum+1,R1
        STF     R1,@EveList2+3  ;The arrival time of part A is passed by Microprocessor 1
        STF     R1,@EveList3+3  ;Schedule the initial departure time to be infinite
        RETS

*-------------------------------------------------------------------------
* FUNCTION DEF : _timing
* schedule the next event, synchronize and advance the simulation clock
* Input        : None
* Register     : R0, R1, R2, R3, R4, R5, AR0, AR1, DP, RC, RS, RE
* Output       : None
*-------------------------------------------------------------------------
_timing:
        LDI     0,R0            ;NOP
LT1     LDP     SharedMem       ;Point to shared memory
        LDI     @PartArr,R0
        CMPI    1,R0            ;Synchronize with microprocessor 1, check for new part A arrival
        BNZ     LT5
        LDI     @M1_M2Dev,R0    ;Event List 2 = M1_M2Dev
        LDI     @M1_M2Dev+1,R1
        LDI     @M1_M2Dev+2,R2
```

```
        LDF     @M1_M2Dev+3,R3
        LDP     _main           ;Point to the data segment
        STI     R0,@EveList2
        STI     R1,@EveList2+1
        STI     R2,@EveList2+2
        STF     R3,@EveList2+3
        LDP     SharedMem       ;Point to shared memory
        LDI     @M1_M2Par,R0    ;WIP queue 0 = M1_M2Par
        LDI     @M1_M2Par+1,R1
        LDI     @M1_M2Par+2,R2
        LDF     @M1_M2Par+3,R3
        LDF     @M1_M2Par+4,R4
        LDF     @M1_M2Par+5,R5
        LDP     _main           ;Point to data segment
        STI     R0,@WIPQ0
        STI     R1,@WIPQ0+1
        STI     R2,@WIPQ0+2
        STF     R3,@WIPQ0+3
        STF     R4,@WIPQ0+4
        STF     R5,@WIPQ0+5
        LDP     SharedMem       ;Point to shared memory
        LDI     0,R0
        STI     R0,@PartArr     ;Inform M1 message read
        STI     R0,@M2Finish    ;Event has not been processed
;Determine the minimum time among event list and set the next event type
;If two event happen at the same time, handle machine broken event first, then
;departure event, and the last is arrival event
LT5     LDP     _main           ;Restore point to data segment
        LDF     @BigNum,R1      ;A initial min time
        LDI     0,R0
        STI     R0,@NxtEvet     ;Next event type = 0
        LDI     1,R0            ;Iteration index
        LDI     @EveList,AR0
        ADDI    3,AR0           ;Point to the first Event List E_Time
LT2     CMPF    *++AR0(4),R1    ;Event List[i].E_Time <= min_time?
        BLT     LT3
        LDF     *AR0,R1         ;Minimum time is
        STI     R0,@NxtEvet     ;Next Event No.
LT3     ADDI    1,R0
        CMPI    3,R0            ;Need to compare all 3 events
        BLE     LT2

        LDI     @NxtEvet,R0     ;If event list is empty
        BNZ     LT4
        LDI     1,R0            ;Exit(1), event list is empty
        STI     R0,@SysExit
        CALL    _exit           ;If even list is empty, stop the simulation and exit
LT4     STF     R1,@Time        ;Simulation clock
        CMPF    @EndTime, R1
        BGT     LTE             ;If Time > simulation end time, then jump out
        LDP     SharedMem       ;Point to shared memory
        CMPF    @SafeTime,R1
        BGT     LT1             ;Workstation 2 cannot process event has time stamp
                                ;larger than the SafeTime.
                                ;SafeTime must be provided by microprocessor 1
        LDP     _main           ;Restore to the data segment
        LDF     @LastTime,R0    ;Last event time
        SUBF3   R0,R1,R2        ;Compute time since last event
        STF     R1, @LastTime   ;Save last event time

        LDI     @PNumInQ,AR0    ;Pointer of NumInQ
        LDI     @PSumInQ,AR1    ;Pointer SumInQ
        LDI     1,RC            ;Repeat 2 times
        RPTB    LT6
        FLOAT   *AR0++,R0
        MPYF    R2,R0
        ADDF    *AR1,R0
LT6     STF     R0,*AR1++       ;Update area of number in queues
LTE     RETS

*---------------------------------------------------------------------------
```

```
* FUNCTION DEF : _arrive
* schedule the next arrival event and handle current arrival event
* Input        : None
* Register     : R0, R1, R2, R4, AR0, AR1, RC, RS, RE
* Output       : None
*-----------------------------------------------------------------------
_arrive:
;Check what type of part arrive
        LDI     @NxtEvet,R0
        CMPI    1,R0
        BZ      LA1
        CMPI    2,R0
        BZ      LA2
LA1     LDI     1,R4            ;Part B type is 1, store in R4
        B       LA3
LA2     LDI     0,R4            ;Part A typeis 0, store in R4
;Check to see if workstation 2 is idle
LA3     LDI     @M2+1,R0
        BNZ     LA4             ;M2 is not idle
        LDI     1,R0            ;M2 is idle, set it to busy
        STI     R0,@M2+1
;Load part from queue to machine 2, WIP to M2 is 3->5, Input Queue to M2 is 4->5,
;therefore starting position is PartType+3, destination is 5
        LDI     5,R1            ;Destination
        LDI     R4,R0
        ADDI    3,R0            ;Staring position for robot 2
        CALL    _rerobot        ;Travel time return in R0
        LDF     @Time,R1
        STF     R1,@CurPart+3   ;Entering time
        ADDF    R0,R1
        STF     R1,@CurPart+4   ;Start processing time
        LDI     @PM2,AR0
        ADDI    R4,AR0
        LDF     *+AR0(3),R0     ;M2.P_Time[Partype]
        CALL    _expon
;Schedule a Part departure after processing, considering the travel time
        ADDF    @CurPart+4,R0   ;Considering the travel time of robot
        STF     R0,@EveList3+3  ;Next departure time
        STI     R4,@EveList3+1  ;Next departure part type
        STF     R0,@CurPart+5   ;Current part finished time
;Update Machine 2 busy period
        ADDF    @SumM2,R0
        SUBF    @CurPart+4,R0
        STF     R0,@SumM2
        BRD     LA8
*** B   LA8 ;BRANCH OCCURS
LA4     LDI     @PNumInQ,AR0    ;M2 is not idle
        ADDI3   R4,AR0,AR1
        LDI     1,R1
        ADDI3   R1,*AR1,R2
        STI     R2,*AR1         ;Increase queue length
        LDI     5,R2
        CMPI    *AR0,R2         ;Check if WIP queue full
        BGT     LA5
        LDF     @EveList3+3,R0  ;Next Departure time
        LDP     SharedMem       ;Point to the shared memory
        STI     R1,@M2QFull     ;WIP queue full
        STF     R0,@M2D_Time    ;Inform M1 next part departure time till block released
        LDP     _main           ;Restore to data segment
LA5     LDI     20,R2           ;Check if input queue full
        CMPI    *+AR0(1),R2
        BGE     LA6
        LDI     3,R0
        STI     R0,@SysExit     ;Input queue full error
        CALL    _exit
LA6     LDI     42,R0           ;Input queue not full, add to the tail of the queue
        MPYI3   R0,R4,AR0
        ADDI    @WIPQ,AR0       ;The first position of the queues, InputQ[PartType][0]
        LDI     @PNumInQ,AR1    ;Pointer of the number in queues
        ADDI    R4,AR1          ;NumInQ[PartType]
        LDI     6,R1
```

```
        MPYI3   R1,*AR1,AR2
        ADDI    AR0,AR2         ;InputQ[PartType][NumInQ[PartType]]
        LDI     5,RC            ;6 data to pass
        RPTB    LA7
        LDI     *AR0++,R1
LA7     STI     R1,*AR2++       ;InputQ[PartType][NumInQ[PartType]] = InputQ[PartType][0]
;If Part A arrive, schedule next Part A to be infinite
;If Part B arrive, schedule next Part B arrival
LA8     CMPI    0,R4
        BZ      LA9             ;Part A arrival
        CMPI    1,R4
        BZ      LA10            ;Part B arrival
LA9     LDP     SharedMem
        LDI     1,R1
        STI     R1,@M2Finish    ;Indicate the event has been processed
        LDP     _main
        LDF     @BigNum+1,R0
        STF     R0,@EveList2+3  ;Delete the part A arrival event
        B       LA11
*** B   LA11 ;BRANCH OCCURS
LA10    LDF     @MeArriB,R0     ;Schedule next Part B arrival
        CALL    _expon
        ADDF    @Time,R0
        STF     R0,@InputQ0+3
        STF     R0,@EveList1+3
LA11    LDI     @PTotNum,AR0    ;Increase total entering part number
        ADDI    R4,AR0
        LDI     1,R0
        ADDI    *AR0,R0
        STI     R0,*AR0
        RETS

*---------------------------------------------------------------------------
* FUNCTION DEF : _depart
* handle current departure event
* Input        : None
* Register     : R0, R1, R2, R3, R4, R5, AR0, AR1, RC, RS, RE, DP
* Output       : None
*---------------------------------------------------------------------------
_depart:
        LDI     @EveList3+1,R4  ;Current part type in R4
;Update current part stay in system, it is realy partial of part stay, need to plus
;the stay time at machine 1
        LDF     @CurPart+5,R0
        SUBF    @CurPart+3,R0
        LDI     @PSumStay,AR0
        ADDI3   R4,AR0,AR1      ;
        ADDF    *AR1,R0
        STF     R0,*AR1         ;SumStay[PartType] += CurPart.P_Finish - CurPart.P_Enter
        LDI     @PSumPart,AR0   ;Update finished parts
        ADDI3   R4,AR0,AR1
        LDI     1,R0
        ADDI3   R0,*AR1,R1
        STI     R1,*AR1
;Check to see if all queues are empty
        LDI     @NumInQ,R1
        BNZ     LD1             ;WIP queue is not empty
        LDI     @NumInQ+1,R2
        BNZ     LD1             ;Input queue is not empty
        LDI     0,R3            ;All queues are empty
        STI     R3,@M2+1        ;Set Machine 2 idle
        LDP     SharedMem
        STI     R3,@M2QFull     ;M2 queues are not full
        LDP     _main
        STI     R3,@EveList3+1  ;Part type does not matter
        LDF     @BigNum+1,R3
        STF     R3,@EveList3+3  ;Set next departure event infinite
        B       LDE
*** B   LDE  ;BRANCH OCCURS
;Queues are not empty, decide next part to be processed according to the dispatching rule.
;Here FIFO is adopted
```

```
LD1     LDI     @NumInQ,R1
        BNZ     LD2
        LDI     1,R4            ;If WIP queue is empty, next part is B
        B       LD5
LD2     LDI     @NumInQ+1,R2
        BNZ     LD3
        LDI     0,R4            ;If Input queue is empty, next part is A
        B       LD5
LD3     LDF     @WIPQ0+9,R3     ;If both are not empty, FIFO
        CMPF    @InputQ0+9,R3   ;if WIPQ[1].P_Enter <= INPUTQ[1].P_Enter
        BGT     LD4             ;If part B arrival first
        LDI     0,R4            ;Part A arrive first
        B       LD5
LD4     LDI     1,R4            ;Part B arrive first
LD5     LDI     1,R0
        STI     R0,@M2+1        ;Set Machine 2 busy
        LDI     @PNumInQ,AR1
        ADDI    R4,AR1
        SUBI3   R0,*AR1,R2      ;Reduce queue length
        STI     R2,*AR1
;Reset WIP queue is not full
        LDI     @NumInQ,R1      ;M2QFull = (NumInQ[0]<WIPLimit)?0:1
        CMPI    5,R1
        LDILT   0,R2
        LDIGE   1,R2
        LDP     SharedMem
        STI     R2,@M2QFull
        LDP     _main
;Update delay accumulator
        LDI     42,R3
        MPYI3   R3,R4,AR0       ;Index position
        ADDI    @WIPQ,AR0       ;Head of queues
        LDI     6,R0
        ADDI3   R0,AR0,AR1      ;First position of queues
        LDI     3,R0
        ADDI3   R0,AR1,AR0      ;First Part in queue Entering time
        LDF     @Time,R5        ;Current simulation clock
        SUBF    *AR0,R5         ;Delay time = Time - P_Enter
        LDI     @PSumDelay,AR0
        ADDI    R4,AR0
        ADDF3   R5,*AR0,R3      ;Sum of delay
        STF     R3,*AR0
;Current Part
        LDI     @PCurPart,AR0   ;Current part pointer in AR0, AR1 points to the first in queue
        LDI     5,RC
        RPTB    LD6
        LDI     *AR1++,R3
LD6     STI     R3,*AR0++       ;CurPart = InputQ[PartType][1]
;Maintain the queue
        LDI     42,R3
        MPYI3   R3,R4,AR0       ;Index position
        ADDI    @WIPQ,AR0       ;Head of queues
        LDI     6,R0
        ADDI    R0,AR0          ;First in queues
        ADDI3   R0,AR0,AR1      ;Next in queue

        LDI     @PNumInQ,AR2    ;Point to the NumInQ
        ADDI    R4,AR2
        LDI     *AR2,R0
        ADDI    1,R0            ;Fill the last position with zeros
        MPYI    6,R0            ;The total number need to be relocated
        SUBI    1,R0
        LDI     R0,RC
        RPTB    LD7
        LDI     *AR1++,R2
LD7     STI     R2,*AR0++
;Schedule next departure
        LDI     6,R0
        LDI     5,R1            ;Parts leave machine 2 and exit the FMS
        CALL    _rerobot        ;Request robot
        ADDF    @Time,R0
```

```
        STF     R0,@CurPart+4   ;Starting time
        LDI     @PM2,AR0
        ADDI3   R4,AR0,AR1
        LDF     *+AR1(3),R0     ;Processing time
        CALL    _expon
        ADDF    @CurPart+4,R0
        STF     R0,@EveList3+3  ;Schedule next departure
        STI     R4,@EveList3+1  ;Event related part type
        STF     R0,@CurPart+5   ;Current part finished time
        LDP     SharedMem       ;Next departure time
        STF     R0,@M2D_Time    ;Inform M1 next part departure time till block released
        LDP     _main           ;
;Update Machine busy period
        ADDF    @SumM2,R0
        SUBF    @CurPart+4,R0
        STF     R0,@SumM2
LDE     RETS

*---------------------------------------------------------------------------
* FUNCTION DEF : _rerobot
* handle robot request, return robot travel time.
* Input        : R0,initial position, R1, destination position
* Register     : R0, R1, AR0, IR0, IR1, (only AR0,IR0,IR1 restored)
* Output       : R0 travel time
*---------------------------------------------------------------------------
_rerobot:
;Calculate travel time, robot need to travel from current position to initial position,
;then from initial position to the destination
        PUSH    AR0
        PUSH    IR0
        PUSH    IR1

        LDI     @Grephon+1,IR0
        MPYI    7,IR0
        ADDI    R0,IR0                  ;From current position to starting position
        LDI     @RNetwork,AR0
        LDI     R0,IR1
        MPYI    7,IR1
        ADDI    R1,IR1                  ;From starting position to destination
        ADDF3   *+AR0(IR1),*+AR0(IR0),R0;Robot travel time
        STI     R1,@Grephon+1           ;Next robot position
        LDF     @Time,R1
        CMPF    @Grephon+2,R1
        BGE     LR1
        ADDF    @Grephon+2,R0           ;Robot is occupied, includes some waiting time
        SUBF    R1,R0
;Next time robot stay at initial position and available time is current time plus travel time
LR1     ADDF    R0,R1                   ;Next available time
        STF     R1,@Grephon+2
        LDF     2.0,R1
        MPYF    R0,R1
        ADDF    @Grephon+3,R1           ;Update robot utilization
        STF     R1,@Grephon+3

        POP     IR1
        POP     IR0
        POP     AR0

        RETS

*---------------------------------------------------------------------------
* FUNCTION DEF : _report
* report simulation results
* Input        : None
* Register     : R0
* Output       : None
*---------------------------------------------------------------------------
_report:

        LDF     @EndTime,R0
        CALL    INV_F30
```

```
        RND     R0,R5           ;R5=1/Time

        LDF     @SumM2,R0
        MPYF    R5,R0
        STF     R0,@AveM2       ;Utilization of M1

        LDF     @Grephon+3,R0
        MPYF    R5,R0
        STF     R0,@AveR        ;Utilization of robot

        LDI     0,R4
LRE1    LDI     @PSumInQ,AR0
        ADDI    R4,AR0
        LDF     *AR0,R0
        MPYF    R5,R0
        LDI     @PAveInQ,AR0
        ADDI    R4,AR0
        STF     R0,*AR0         ;Average in queues

        LDI     @PSumPart,AR0
        ADDI    R4,AR0
        FLOAT   *AR0,R0
        CALL    INV_F30
        RND     R0,R2           ;R2 = 1/SumPart[i]

        LDI     @PSumDelay,AR0
        ADDI    R4,AR0
        LDF     *AR0,R0
        MPYF    R2,R0
        LDI     @PAveDelay,AR0
        ADDI    R4,AR0
        STF     R0,*AR0         ;Average delay

        LDI     @PSumStay,AR0
        ADDI    R4,AR0
        LDF     *AR0,R0
        MPYF    R2,R0
        LDI     @PAveStay,AR0
        ADDI    R4,AR0
        STF     R0,*AR0         ;Average stay

        ADDI    1,R4
        CMPI    2,R4
        BLT     LRE1
        RETS

*-----------------------------------------------------------------------
* FUNCTION DEF : _exit
* Unusually exit
* Input        : none
* Register     : none
* Output       : none
*-----------------------------------------------------------------------
_exit:
EXITSIM B       $

*-----------------------------------------------------------------------
* FUNCTION DEF : _barrier
* Unusually exit
* Input        : none
* Register     : R0
* Output       : none
*-----------------------------------------------------------------------
_barrier:
        LDP     SharedMem
        LDI     @Barrier1,R0
        ADDI    1,R0
        STI     R0,@Barrier1
B1      LDI     @Barrier1,R0
        CMPI    2,R0
        BLT     B1
```

```
        LDP     @_main
        RETS

*---------------------------------------------------------------------
* FUNCTION DEF : _expon
* Generate a random number with exponential distribution
* Input        : R0 is the mean
* Register     : R1
* Output       : R0 is the random number with exponential distribution
*---------------------------------------------------------------------
_expon:
*Fast 32 bit uniform random number generator
        PUSH    R1
        PUSHF   R1

        NEGF    R0, R1          ; R1 is the negative of the mean
        PUSH    R1
        PUSHF   R1              ; Save -mean in the stack, all 40 bits
        LDF     *AR4++(1),R0    ; Get a random number
        POPF    R1              ; -Mean
        POP     R1
        MPYF    R1, R0          ; random number =  -mean*ln(u)

        POPF    R1
        POP     R1
        RETS

*---------------------------------------------------------------------
* FUNCTION DEF : _M2lnUni
* Generate a random number with exponential distribution
* Input        : None
* Register     : R0
* Output       : R0 Is the ln(uniform) random number series
*---------------------------------------------------------------------
_M2lnUni:
        LDI     @M2SEED,R0      ; Call here for last SEED
        MPYI    @A,R0           ; Calculate r = (A*r+C)%m
        ADDI    @C,R0           ;
        AND     @MASK1, R0      ; m=2^23
        STI     R0,@M2SEED      ; Result is returned
        ADDI    1,R0            ;
        FLOAT   R0,R0           ;
        MPYF    @INVE23,R0      ; r/m
        CALL    _ln             ; ln(u)
        RETS
```

Note:

1. Subroutines _ln and _INVF30 are the same as in 1Mhard1.asm
2. Only one set of programs is included here

C.2.3 Sample Data Recorded from the Prototype Simulator

Table A5 and A6 show two set of data recorded for simulating 2-workstation case. There are total 15 sets of such data generated by randomly changing seeds.

A5. Experiment No. 1 for simulating 2-workstaton (seeds 0/0)

Experiment Data for 2-Machine Simulation **Date: Feb. 23** **Experiment No.: 1**					
		PDESM		**HARD**	
Executed Cycles:		671163		582855	
Simulation Time:		14405 (14408)		14405 (14408)	
Seed:		0	0	0	0
		Lathe 1 (M1)	**Mill 1 (M2)**	**Lathe 1 (M1)**	**Mill 1 (M2)**
MTBA	Part A	15.0	N/A	15.0	N/A
	Part B	N/A	20.0	N/A	20.0
No. Part Arrivals	Part A	962	953	962	953
	Part B	N/A	710	N/A	710
No.Part Processed	Part A	953	948	953	948
	Part B	N/A	704	N/A	704
Avg in Que	WIP	N/A	PA 1.445	N/A	PA 1.445
	Input	PA 0.971	PB 1.152	PA 0.971	PB 1.152
Util of Mach.		0.402	0.701	0.402	0.701
Mach. Blocked		0.061	N/A	0.061	N/A
Util of Robot		R1 0.079	R2 0.154	R1 0.079	R2 0.154
Avg. Stay on Mach.	Part A	22.104	30.266	22.104	30.266
	Part B	N/A	27.586	N/A	27.586
Avg. Delay in Que	WIP	N/A	PA 21.687	N/A	PA 21.687
	Input	PA 14.344	PB 23.360	PA 14.344	PB 23.360

A6 Experiment No.15 for simulating 2-workstaton (seeds 702749/431012)

Experiment Data for 2-Machine Simulation **Date: Feb. 23** **Experiment No.: 15**						
		PDESM			**HARD**	
Executed Cycles:		678891			623771	
Simulation Time:		14404 (14401)			14404 (14401)	
Seed:		702749	431012		702749	431012
		Lathe 1 (M1)	**Mill 1 (M2)**		**Lathe 1 (M1)**	**Mill 1 (M2)**
MTBA	Part A	15.0	N/A		15.0	N/A
	Part B	N/A	20.0		N/A	20.0
No. Part Arrivals	Part A	1018	1016		1017	1016
	Part B	N/A	709		N/A	709
No.Part Processed	Part A	1016	1014		1016	1014
	Part B	N/A	704		N/A	704
Avg in Que	WIP	N/A	PA 2.226		N/A	PA 2.226
	Input	PA 1.577	PB 1.729		PA 1.577	PB 1.729
Util of Mach.		0.435	0.770		0.435	0.770
Mach. Blocked		0.114	N/A		0.114	N/A
Util of Robot		R1 0.085	R2 0.154		R1 0.085	R2 0.154
Avg. Stay on Mach.	Part A	30.869	40.470		30.869	40.470
	Part B	N/A	39.937		N/A	39.937
Avg. Delay in Que	WIP	N/A	PA 31.578		N/A	PA 31.578
	Input	PA 22.345	PB 35.335		PA 22.345	PB 35.335

Table A7 and A8 show two set of data recorded for simulating 2-workstation case. There are total 9 sets of such data generated by changing arrival pattern.

A7 Experiment No.1 for simulating 2-workstaton (arrival 15/15)

Experiment Data for 2-Machine Simulation **Date: Feb. 24** **Experiment No.: 1**						
		PDESM			**HARD**	
Executed Cycles:		794444			678112	
Simulation Time:		14404 (14404)			14404 (14404)	
Seed:		0	0		0	0
		Lathe 1 (M1)	**Mill 1 (M2)**		**Lathe 1 (M1)**	**Mill 1 (M2)**
MTBA	Part A	15.0	N/A		15.0	N/A
	Part B	N/A	15.0		N/A	15.0
No. Part Arrivals	Part A	980	974		980	974
	Part B	N/A	945		N/A	945
No.Part Processed	Part A	974	969		975	969
	Part B	N/A	939		N/A	939
Avg in Que	WIP	N/A	PA 2.123		N/A	PA 2.123
	Input	PA 1.566	PB 2.306		PA 1.566	PB 2.306
Util of Mach.		0.416	0.776		0.416	0.776
Mach. Blocked		0.114	N/A		0.114	N/A
Util of Robot		R1 0.081	R2 0.168		R1 0.081	R2 0.168
Avg. Stay on Mach.	Part A	31.648	39.872		31.689	39.872
	Part B	N/A	39.516		N/A	39.516
Avg. Delay in Que	WIP	N/A	PA 31.403		N/A	PA 31.403
	Input	PA 23.035	PB 35.147		PA 23.080	PB 35.147

A8 Experiment No.9 for simulating 2-workstaton (arrival 25/25)

Experiment Data for 2-Machine Simulation **Date: Feb. 24** **Experiment No.: 9**					
		PDESM		**HARD**	
Executed Cycles:		387496		371195	
Simulation Time:		14418 (14418)		14496 (14418)	
Seed:		0	0	0	0
		Lathe 1 (M1)	**Mill 1 (M2)**	**Lathe 1 (M1)**	**Mill 1 (M2)**
MTBA	Part A	25.0	N/A	25.0	N/A
	Part B	N/A	25.0	N/A	25.0
No. Part Arrivals	Part A	570	569	570	569
	Part B	N/A	573	N/A	573
No.Part Processed	Part A	569	569	569	569
	Part B	N/A	571	N/A	571
Avg in Que	WIP	N/A	PA 0.322	N/A	PA 0.322
	Input	PA 0.158	PB 0.326	PA 0.158	PB 0.326
Util of Mach.		0.229	0.483	0.231	0.483
Mach. Blocked		0.001	N/A	0.001	N/A
Util of Robot		R1 0.047	R2 0.116	R1 0.047	R2 0.116
Avg. Stay on Mach.	Part A	10.625	17.186	10.625	17.186
	Part B	N/A	12.842	N/A	12.842
Avg. Delay in Que	WIP	N/A	PA 8.141	N/A	PA 8.141
	Input	PA 3.993	PB 8.230	PA 3.993	PB 8.230

C.3 Experiment Case for 4-workstation Manufacturing Cell

C.3.1 Simulation Results from ProModel

```
--------------------------------------------------------------------------------
General Report
Output from C:\Project\FMSModel\ThesisModel\Fms-4m2r.mod [DesktopFMS]
Date: Apr/14/2001   Time: 09:05:37 AM
--------------------------------------------------------------------------------
Scenario        : Normal Run
Replication     : Average
Period          : Final Report (0 sec to 240 hr Elapsed: 240 hr)
Simulation Time : 240 hr
--------------------------------------------------------------------------------

LOCATIONS
                                        Average
Location  Scheduled            Total    Minutes    Average   Maximum   Current
Name           Hours Capacity  Entries  Per Entry   Contents  Contents  Contents  % Util
----------- ------  --------  -------  ---------  ---------  --------  --------  ------
Lathe1         240         1    909.5   7.057815   0.445714         1       0.5   44.57  (Average)
Lathe1           0         0  25.3563   0.369497   0.025663         0  0.527046    2.57  (Std. Dev.)
Mill1          240         1     1624   6.349654   0.716011         1       0.6   71.60  (Average)
Mill1            0         0  36.1417   0.093313  0.0148251         0  0.516398    1.48  (Std. Dev.)
Conveyor21     240        20    948.9  10.136704   0.666923        10       0.1    3.33  (Average)
Conveyor21       0         0  29.6964   4.163780   0.274974   2.70801  0.316228    1.37  (Std. Dev.)
Conveyor22     240        20    718.3  21.882652    1.09078       8.6       1.3    5.45  (Average)
Conveyor22       0         0  22.8378   2.386476   0.115859   1.64655   1.88856    0.58  (Std. Dev.)
WIPQ1          240         5      909  24.216268    1.52881         5         2   30.58  (Average)
WIPQ1            0         0  25.5256   2.092649   0.141945         0   1.94365    2.84  (Std. Dev.)
Lathe2         240         1    988.1   7.809491   0.536244         1       0.5   53.62  (Average)
Lathe2           0         0  20.2509   0.544268  0.0444913         0  0.527046    4.45  (Std. Dev.)
Mill2          240         1   1710.6   6.527740   0.775414         1       0.9   77.54  (Average)
Mill2            0         0  39.3762   0.126496  0.0224339         0  0.316228    2.24  (Std. Dev.)
Cov3           240        20    990.2  17.193481    1.18772      12.6       2.1    5.94  (Average)
Cov3             0         0  21.7603   6.624019   0.472997   3.62706   4.09471    2.36  (Std. Dev.)
Cov4           240        20    726.2  26.978491      1.368      10.7         1    6.84  (Average)
Cov4             0         0  35.9036   5.617303   0.327949   2.58414  0.942809    1.64  (Std. Dev.)
WIPQ2          240         5    987.6  30.029841    2.06252         5       2.2   41.25  (Average)
WIPQ2            0         0  20.4135   3.981191   0.300678         0   1.93218    6.01  (Std. Dev.)


LOCATION STATES BY PERCENTAGE (Multiple Capacity)
                                    %       |
Location    Scheduled       %  Partially      % |    %
Name            Hours   Empty   Occupied   Full | Down
----------  ---------  ------  ---------  ----- | ----
Conveyor21        240   71.95      28.05   0.00 | 0.00  (Average)
Conveyor21          0    3.26       3.26   0.00 | 0.00  (Std. Dev.)
Conveyor22        240   52.01      47.99   0.00 | 0.00  (Average)
Conveyor22          0    1.95       1.95   0.00 | 0.00  (Std. Dev.)
WIPQ1             240   43.86      44.93  11.20 | 0.00  (Average)
WIPQ1               0    3.27       3.40   2.78 | 0.00  (Std. Dev.)
Cov3              240   62.69      37.31   0.00 | 0.00  (Average)
Cov3                0    6.45       6.45   0.00 | 0.00  (Std. Dev.)
Cov4              240   45.57      54.43   0.00 | 0.00  (Average)
Cov4                0    5.27       5.27   0.00 | 0.00  (Std. Dev.)
WIPQ2             240   33.46      46.97  19.57 | 0.00  (Average)
WIPQ2               0    5.21       2.62   5.41 | 0.00  (Std. Dev.)


LOCATION STATES BY PERCENTAGE (Single Capacity/Tanks)

Location     Scheduled          %      %       %        %        %     %
Name             Hours  Operation  Setup    Idle  Waiting  Blocked  Down
-----------  ---------  ---------  -----  ------  -------  -------  ----
Lathe1             240      38.02   0.00   50.05     1.20     5.36  5.38  (Average)
Lathe1               0       1.62   0.00    2.57     0.07     1.73  0.00  (Std. Dev.)
Mill1              240      70.63   0.00   28.40     0.97     0.00  0.00  (Average)
Mill1                0       1.44   0.00    1.48     0.05     0.00  0.00  (Std. Dev.)
Lathe2             240      41.22   0.00   46.37     2.29    10.11  0.00  (Average)
```

Lathe2	0	1.32	0.00	4.45	0.15	3.18	0.00	(Std. Dev.)
Mill2	240	75.51	0.00	22.46	2.03	0.00	0.00	(Average)
Mill2	0	2.15	0.00	2.24	0.13	0.00	0.00	(Std. Dev.)

RESOURCES

Resource Name	Units	Scheduled Hours	Number Of Times Used	Average Minutes Per Usage	Average Minutes Travel To Use	Average Minutes Travel To Park	% Blocked In Travel	% Util	
Gryphon1	1	240	5105.2	0.445818	0.383872	0.000000	0.00	29.40	(Average)
Gryphon1	0	0	117.816	0.001119	0.001501	0.000000	0.00	0.35	(Std. Dev.)
Gryphon2	1	240	5396	0.500090	0.425528	0.000000	0.00	34.68	(Average)
Gryphon2	0	0	102.461	0.000035	0.004170	0.000000	0.00	0.43	(Std. Dev.)

RESOURCE STATES BY PERCENTAGE

Resource Name	Scheduled Hours	% In Use	% Travel To Use	% Travel To Park	% Idle	% Down	
Gryphon1	240	15.80	13.60	0.00	70.60	0.00	(Average)
Gryphon1	0	0.18	0.18	0.00	0.35	0.00	(Std. Dev.)
Gryphon2	240	18.74	15.94	0.00	65.32	0.00	(Average)
Gryphon2	0	0.18	0.27	0.00	0.43	0.00	(Std. Dev.)

ENTITY ACTIVITY

Entity Name	Total Exits	Current Quantity In System	Average Minutes In System	Average Minutes In Move Logic	Average Minutes Wait For Res, etc.	Average Minutes In Operation	Average Minutes Blocked	
PartA1	906.5	3.1	50.651325	1.733904	21.070015	14.386111	13.461295	(Average)
PartA1	24.0197	1.96921	5.683540	0.014573	4.674443	0.228465	0.972389	(Std. Dev.)
PartB1	716.9	1.4	26.645318	1.363541	12.233673	4.052614	8.995490	(Average)
PartB1	23.0336	2.17051	2.366611	0.007581	2.251824	0.090147	0.306665	(Std. Dev.)
PartA2	985.1	5.1	64.538049	2.483931	32.238407	14.136429	15.679282	(Average)
PartA2	20.8457	5.08702	10.558698	0.053584	8.618813	0.246064	1.822939	(Std. Dev.)
PartB2	724.6	1.6	31.531889	1.092450	16.192687	3.945331	10.301421	(Average)
PartB2	35.8584	1.26491	5.688879	0.010743	5.156164	0.121491	0.732455	(Std. Dev.)

ENTITY STATES BY PERCENTAGE

Entity Name	% In Move Logic	% Wait For Res, etc.	% In Operation	% Blocked	
PartA1	3.46	41.17	28.68	26.69	(Average)
PartA1	0.35	4.29	2.82	1.30	(Std. Dev.)
PartB1	5.15	45.56	15.32	33.97	(Average)
PartB1	0.47	4.53	1.42	2.77	(Std. Dev.)
PartA2	3.93	49.19	22.40	24.48	(Average)
PartA2	0.56	5.27	3.42	1.42	(Std. Dev.)
PartB2	3.56	50.27	12.82	33.34	(Average)
PartB2	0.61	6.95	1.95	4.54	(Std. Dev.)

VARIABLES

Variable Name	Total Changes	Average Minutes Per Change	Minimum Value	Maximum Value	Current Value	Average Value	
Total Parts Num A	1899.8	7.579395	0	1899.8	1899.8	950.082	(Average)
Total Parts Num A	41.0008	0.162627	0	41.0008	41.0008	31.8679	(Std. Dev.)

Total Parts Num B	1444.5	9.965879	0	1444.5	1444.5	730.861	(Average)
Total Parts Num B	34.6707	0.233608	0	34.6707	34.6707	18.7378	(Std. Dev.)
Total Routed	39.3	363.806063	0	39.3	39.3	19.4839	(Average)
Total Routed	8.76926	85.427126	0	8.76926	8.76926	5.00481	(Std. Dev.)

C.3.2 Assembly Programs

C.3.2.1 Code running on DSP 1

```
*-----------------------------------------------------------------------
* 4MHardL1.ASM -- parallel discrete event simulation
* of a scenario: 4 machine, two part types, two robots, and considering machine breakdown
* This is the program running on Microprocessor 1, using to simulating Lathe 1
* Programmed by: Dong Xu
* Date         : March 22, 2001
* Modified     : Apr. 2, 2001
*-----------------------------------------------------------------------
        .start "data", 0x809900     ;Data section, 809900-8099FF
        .start "code", 0x809A00     ;Code section, 809A00-809EFF
        .start "isrcode", 0x809D00  ;Interrupt service routine section
        .start "shared1", 0xA00000  ;Shared memory section 1, A00000-A003FF
        .start "shared2", 0xC00000  ;Shared memory section 2, C00000-C003FF
        .start "text", 0xB00000     ;text section, B00000-B07FFF

        .sect "data"
;Constants for Timer 0 control
EINT    .word 00002000h    ;Enable global interrupt
ETINT0  .word 00000100h    ;Enable Timer0 interrupt
PERIOD  .int 100000        ;Timer period
BISR    .word 60809D00h    ;Branch to interrupt service routine, 60 jump to 0x809D00h
COUNT   .int 0             ;Count of Timer interrupt
TOTCYC  .int 0             ;Total cycles
SysExit .int 5             ;Exit state of the simulation,
;                            0 success, 1 event list empty, 2 Lathe 1 input queue overflow
;                            3 Mill 1 input queue overflow, 5 other fatal error, 6 ln negative

;These global variables are used to pass parameters between functions to execute the simulation
;Average of simulation results
AveInQ      .float 0.0  ;Average of number parts in queue
AveM1       .float 0.0  ;Average of machine 1 utilization
AveBlock    .float 0.0  ;Average block time of parts
AveR        .float 0.0  ;Average of robot utilization
AveStay     .float 0.0  ;Average time stay in system
AveDelay    .float 0.0  ;Average delay time of parts

NumInQ      .int 0      ;Number of parts waiting in M1 input queue
TotNum      .int 1      ;Total number of part entering
SumPartA    .int 0      ;Total finished Part A
M1PState    .int 0      ;Previous state of machine 1 before broken

SumInQ      .float 0.0  ;Sum of number parts in queue
SumM1       .float 0.0  ;Sum of machine 1 utilization
SumDelay    .float 0.0  ;Total delay time of parts
SumBlock    .float 0.0  ;Total block time of parts
SumStay     .float 0.0  ;Total time stay in system
EndTime     .float 14400.0;End simulation time, 240 hours
MeArriA     .float 15.0 ;Mean time between arrival of Part A

Time        .float 0.0  ;Current simulating time, in minutes
LastTime    .float 0.0  ;Time of last event
NxtEvet     .int 0      ;Next event type, 0=empty, 1=arrival, 2=departure, 3=machine broken
;Event list
;current event
EveList0    .int 0      ;Event type = 0(current), 1(Arrival), 2(Departure), 3(send) 4(Machine
Broken)
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 1      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
;arrival event
EveList1    .int 1      ;Event type
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 1      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
;departure event
EveList2    .int 2      ;Event type
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
```

```
            .int 1      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
;Part A sending to Mill 1 event
EveList3    .int 3      ;Event type
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 1      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
;Workstation 1 breakdown event
EveList4    .int 4      ;Event type
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 1      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
;Part A routing event
EveList5    .int 5      ;Event type
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 1      ;Which machine related to the event, 1(Lathe 1), 2(Mill 1)
            .float 0.0  ;Event happen time

InputQ0     .space 132  ;Input queue for the workstation 1, =6*21+6 for tail

;Defined the structure used in the program
;Describe workstation 1
M1          .int 1      ;Machine type, 1 = lathe, 2 = mill
            .int 0      ;Machine state = 0(idle), 1(busy), 2(blocked), 3(broken)
            .int 0      ;Current operating part type, 0 = Part A, 1 = Part B
            .float 6.0  ;Processing time for Part A
            .float 0.0  ;Processing time for Part B, not available
            .float 450  ;Mean time between failure
            .float 25   ;Mean time to repair

;Current part scheduled to be processed on machine
CurPart     .int 0      ;Part type = 0(Part A), 1(Part B)
            .int 1      ;Operation required on workstation 1, 0(No), 1(Yes)
            .int 1      ;Operation required on workstation 2, 0(No), 1(Yes)
            .float 0.0  ;Time when entering the FMS
            .float 0.0  ;Processing start time to one workstation
            .float 0.0  ;Finish time of one operation on one workstation

;Define the pointers to the arrays
PEveList    .word EveList0
PInputQ     .word InputQ0
PRNetwork   .word RNetwork1
PCurPart    .word CurPart
PM1_M2Part  .word M1_M2Part
PM1_M2Dev   .word M1_M2Dev
PGrephon    .word Grephon
PM1_M3Part  .word M1_M3Part
PSharedMem1 .word M2Finish
PSharedMem2 .word M1Down

*-----------------------------------------------------------------------------
* DEFINE CONSTANTS
*-----------------------------------------------------------------------------
QLimit      .int 20             ;Length limit of the queue
WIPLimit    .int 5              ;Limit of WIP queue
NumEvet     .int 5              ;Number of event
NumPType    .int 2              ;Number of part type

;Constant used by random number generation subroutine
A       .int    1078373         ; Constants needed for RAND
C       .int    2311527         ;
SEED    .int    0          ;
MASK    .word   0xFF7FFFFF      ;Mask for fast inverse float
MASK1   .word   0x007FFFFF      ;Mask for exponential distribution, %2^23
INVE23  .word   0xE9000000      ;1/2^23
BigNum  .float  1.0e29          ;Constant big value
        .float  1.0e30          ;

;Internal constants for ln(u)
;Scaling coefficients for ln(1+x)
LNRM    .float  0.6931471806    ; LN(2)
```

```
C0      .float  1.0000000000     ; C0 (1.0)
;       Polynomial coefficients for ln(1+X), 0 <= X < 1.
        .float  0.9999964239     ; TOP OF C1
        .float  -0.4998741238    ; TOP OF C2
        .float  0.3317990258     ; TOP OF C3
        .float  -0.2407338084    ; TOP OF C4
        .float  0.1676540711     ; TOP OF C5
        .float  -0.0953293897    ; TOP OF C6
        .float  0.0360884937     ; TOP OF C7
C8      .float  -0.0064535442    ; TOP OF C8
AC8     .word   C8

*-------------------------------------------------------------------------------
*Special signals for synchronization purpose, should be stored in the shared memory section 1,
*accessed by to both Microprocessor 1 and 2
*-------------------------------------------------------------------------------
    .sect "shared1"
M2Finish    .int 1      ;Indicate M2 finish the previous part departure event, *AR6
M2QFull     .int 0      ;Signal of WIP queue, 0 = not full, 1 = full, *+AR6(1)
PartArr     .int 0      ;Indicate Part A leave from Machine 1 to 2, *+AR6(2)
M2SafeTime  .float 0.0  ;Indicate how far simulation clock on M2 can advance, *+AR6(3)
M2D_Time    .float 0.0  ;Scheduled next part departure time on machine 2, *+AR6(4)

;Departure message from Machine 1 to Machine
M1_M2Dev    .int 0      ;Event type = 0(current), 1(Arrival), 2(Departure), 3(Machine Broken)
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 1      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
;Current Part transported from Machine 1 to Machine 2
M1_M2Part   .int 0      ;Part type = 0(Part A), 1(Part B)
            .int 1      ;Operation required on workstation 1, 0(No), 1(Yes)
            .int 1      ;Operation required on workstation 2, 0(No), 1(Yes)
            .float 0.0  ;Time when entering the FMS
            .float 0.0  ;Arrival time to one workstation
            .float 0.0  ;Finish time of one operation on one workstation

;Describe the robot
Grephon     .int 0      ;Robot state = 0(idle), 1(occupied)
            .int 1      ;current position of robot
            .float 0.0  ;Next available time of the robot
            .float 0.0  ;Next available time of the robot
            .float 0.0  ;Area of robot utilization

;Robot 1 path network, using real travel time if necessary
RNetwork1   .float 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
            .float 0.0, 0.0, 0.3, 0.6, 0.9, 1.2, 1.5, 0.3
            .float 0.0, 0.3, 0.0, 0.3, 0.6, 0.9, 1.2, 0.3
            .float 0.0, 0.6, 0.3, 0.0, 0.3, 0.6, 0.9, 0.6
            .float 0.0, 0.9, 0.6, 0.3, 0.0, 0.3, 0.6, 0.6
            .float 0.0, 1.2, 0.9, 0.6, 0.3, 0.0, 0.3, 0.9
            .float 0.0, 1.5, 1.2, 0.9, 0.6, 0.3, 0.0, 0.9
            .float 0.0, 0.3, 0.3, 0.6, 0.6, 0.9, 0.9, 0.0

SharedMem1  .word M2Finish
Barrier1    .int 0      ;Barrier counter to start the process

*-------------------------------------------------------------------------------
*Special signals for synchronization purpose, should be stored in the shared memory section 2,
*accessed by to both Microprocessor 1 and 3
*-------------------------------------------------------------------------------
    .sect "shared2"
;Flags to synchronize the simulation of M1 and M2, indexed by AR5
M1Down      .int 0      ;Indicate Lathe 1 down, *AR5
PartArrival .int 0      ;Routed part A read by M2, 0=not, 1=read,2=no part, *+AR5(1)
PermitSend  .int 0      ;Part A allowed by M2 to be routed, 0=not, 1= allowed, *+AR5(2)
RequestSend .int 0      ;Request to send part A from Lathe 1 to Lathe 2, 0=no, 1=Yes, *+AR5(3)
M1SafeTime  .float 0.0  ;Indicate leading time on M1 during block, *+AR5(4)
M3SafeTime  .float 0.0  ;Indicate leading time on M3, *+AR5(5)
M3R_Time    .float 0.0  ;Scheduled next part routing time from Lathe 1 to Lathe 2, *+AR5(6)

;Part A transported from Lathe 1 to Lathe 2
```

```
M1_M3Part   .int 0      ;Part type = 0(Part A), 1(Part B)
            .int 1      ;Operation required on Lathe, 0(No), 1(Yes)
            .int 1      ;Operation required on Mill, 0(No), 1(Yes)
            .float 0.0  ;Time when entering the FMS
            .float 0.0  ;Arrival time to one workstation
            .float 0.0  ;Finish time of one operation on one workstation

SharedMem2  .word M1Down
Barrier2    .int 0      ;Barrier counter to start the process

    .sect "code"
    .entry  _main
*-------------------------------------------------------------------------------
* FUNCTION DEF : _main
* Control the advance of the program
*-------------------------------------------------------------------------------
_main:
        LDP     @_main          ;Set Data Segment Pointer
        LDI     @STACK,SP       ;Set stack pointer
        LDI     @PSharedMem2,AR5 ;The index of flags in shared memory section 2, for
Microprocessor 3
        LDI     @PSharedMem1,AR6 ;The index of flags in shared memory section 1, for
Microprocessor 2

        CALL    _initial        ;Initialize the simulation, especially data structure
        CALL    _barrier

;Setup Timer 0 to count the execution time
        LDI     @BISR, R0
        STI     R0, @9FC9h      ;Save the branch instruction to Timer0 interrupt vector
        LDI     200h, R1
        STI     R1, @8020h      ;Using internal H1/2 clock for Timer 0
        LDI     @PERIOD, R1
        STI     R1, @8028h      ;Set period of Timer 0
        LDI     0,R1
        STI     R1,@8024h       ;Reset counter register
        LDI     2C0h, R1
        STI     R1, @8020h      ;Start Timer 0
        OR      @ETINT0, IE     ;Enable Timer 0 interrupt
        OR      @EINT, ST       ;Enable Interrupt

LM1     LDF     @Time,R0
        CMPF    @EndTime,R0     ;Does simulation end?
        BGT     LM2

        CALL    _timing         ;Determine the next event type

;Invoke the appropriate event handle function
        LDI     @PSUB,AR0
        ADDI    @NxtEvet,AR0
        LDI     *AR0,R0
        CALLU   R0              ;Handle part arrival, departure, or machine breakdown events
        B       LM1
LM2
        CALL    _report         ;Report simulation results

        LDI     0,R0
        STI     R0,@SysExit     ;Successfully run

;Calculate the execution time
        LDI     200h, R1
        STI     R1, @8020h      ;Halt the timer
        LDI     @PERIOD, R0     ;Calculate the whole cycles
        MPYI    @COUNT, R0
        LDI     @8024h, R1      ;Current counter number
        ADDI    R1, R0
        STI     R0, @TOTCYC     ;Total cycle used
END     B       $               ;End of the program


*-------------------------------------------------------------------------
```

```
* FUNCTION DEF : _initial
* Fill the robot path network, initialize the data structures and the first arrival
* Input        : None
* Register     : R1, AR0, RC, RS, RE
* Output       : None
*-------------------------------------------------------------------------
_initial:
;Describe the current part on workstation 1, InputQ[0] is the storage of next part
        LDI     0,R1
        STI     R1,@InputQ0     ;Part A
        LDI     1,R1
        STI     R1,@InputQ0+1   ;Part A need turning operation on workstation 1
        STI     R1,@InputQ0+2   ;Part A need drilling operation on workstation 2
        LDF     @MeArriA,R0
        CALL    _expon
        STF     R0,@InputQ0+3  ;Part A first arrival time
        LDF     0.0,R1
        STF     R1,@InputQ0+4   ;Part A start on workstation 1, depending on robot
        STF     R1,@InputQ0+5   ;Time to finish operation on workstation 1 does not known yet

        LDF     @InputQ0+3,R1
        STF     R1,@EveList1+3  ;Schedule the arrive time
        LDF     @M1+5,R1
        STF     R1,@EveList4+3  ;Schedule first machine down
        LDF     @BigNum+1,R1
        STF     R1,@EveList2+3  ;Schedule the initial departure time to be infinite
        STF     R1,@EveList3+3  ;Schedule the initial sending event to be infinite
        STF     R1,@EveList5+3  ;First route event time to be infinite
        RETS

*-------------------------------------------------------------------------
* FUNCTION DEF : _timing
* schedule the next event, synchronize and advance the simulation clock
* Input        : None
* Register     : R0, R1, R2, AR0, DP
* Output       : None
*-------------------------------------------------------------------------
_timing:
;Check if Lathe 1 down
        LDI     @M1+1,R0        ;Find current state of Lathe 1
        CMPI    3,R0            ;If Lathe 1 down?
        BNE     LT5             ;Lathe 1 not down, normal situation
;When Lathe 1 down, considering the routing event
LT1     LDI     *+AR5(2),R0     ;Lathe 1 down situation
        CMPI    1,R0            ;If next Part A sending permitted by M2?
        BNE     LT2             ;No, jump to LT2
        LDF     *+AR5(6),R1     ;Yes, read in the routing time
        LDI     0,R0
        STI     R0,*+AR5(2)     ;Clear the sending next part flag
        STF     R1,@EveList5+3  ;Schedule a part A routing event
;Determine the minimum time among event list and set the next event type
;If two event happen at the same time, handle machine broken event first, then
;departure event, and the last is arrival event
LT2     LDF     @BigNum,R1      ;A initial min time
        LDI     0,R0
        STI     R0,@NxtEvet     ;Next event type = 0
        LDI     1,R0            ;Iteration index
        LDI     @PEveList,AR0
        ADDI    3,AR0           ;Point to the first Event List E_Time
LT3     CMPF    *++AR0(4),R1    ;Event List[i].E_Time <= min_time?
        BLT     LT4
        LDF     *AR0,R1         ;Minimum time is R1
        STI     R0,@NxtEvet     ;Next Event No.
LT4     ADDI    1,R0
        CMPI    5,R0            ;Need to compare all events
        BLE     LT3
;Set safetime and check to see if the event can be safely processed
        LDF     @EveList3+3,R0  ;
        CMPF    @BigNum,R0
        BLT     LT0
        LDF     R1,R0           ;Current Time
```

```
LT0     STF     R0,*+AR6(3)     ;Safe time for M2 is next departure time
                                ;Microprocessor 2 can process events before this M2safetime
        LDF     @EveList4+3,R0
        STF     R0,*+AR5(5)     ;Set M3 safe time to be the next time of broken event

        CMPF    *+AR5(4),R1     ;Is time <= M1 Safe Time?
        BGT     LT1             ;No, recheck the safe time and all
        B       LT8             ;Jump over normal situation

;Under normal situation, do not need to consider the routing event
LT5     LDF     @BigNum,R1      ;A initial min time
        LDI     0,R0
        STI     R0,@NxtEvet     ;Next event type = 0
;Determine the minimum time among event list and set the next event type
;If two event happen at the same time, handle machine broken event first, then
;departure event, and the last is arrival event
        LDI     1,R0            ;Iteration index
        LDI     @PEveList,AR0
        ADDI    3,AR0           ;Point to the first Event List E_Time
GLT1    CMPF    *++AR0(4),R1    ;Event List[i].E_Time <= min_time?
        BLT     GLT2
        LDF     *AR0,R1         ;Minimum time is R1
        STI     R0,@NxtEvet     ;Next Event No.
GLT2    ADDI    1,R0
        CMPI    4,R0            ;Need to compare only 4 events
        BLE     GLT1
        LDI     @NxtEvet,R0     ;If event list is empty
        BNZ     LT8
        LDI     1,R0            ;Exit(1), event list is empty
        CALL    _exit           ;If even list is empty, stop the simulation and exit

;All situations
LT8     LDI     @NxtEvet,R0     ;If event list is empty
        BNZ     LT9
        LDI     1,R0            ;Exit(1), event list is empty
        CALL    _exit           ;If even list is empty, stop the simulation and exit
LT9     STF     R1,@Time        ;Update Simulation clock

        LDF     @LastTime,R0    ;Last event time
        SUBF3   R0,R1,R2        ;Compute time since last event
        STF     R1, @LastTime   ;Save last event time

        LDF     @EveList3+3,R0  ;
        CMPF    @BigNum,R0
        BLT     LT10
        LDF     @Time,R0
LT10    STF     R0,*+AR6(3)     ;Safe time for M2 is next departure time
                                ;Microprocessor 2 can process events before this M2safetime
        LDF     @EveList4+3,R0
        STF     R0,*+AR5(5)     ;Set M3 safe time to be the next time of broken event

        FLOAT   @NumInQ,R0      ;Statistical update
        MPYF    R2,R0
        ADDF    @SumInQ,R0
        STF     R0,@SumInQ      ;Update area of number in M1 queue

        RETS


*-------------------------------------------------------------------------
* FUNCTION DEF : _arrive
* schedule the next arrival event and handle current arrival event
* Input        : None
* Register     : R0, R1, R2, AR0, AR1
* Output       : None
*-------------------------------------------------------------------------
_arrive:
        LDI     @M1+1,R0        ;Check to see if workstation 1 is idle?
        BNZ     LA1             ;If not idle, go to LA1
        LDI     1,R0            ;If idle
```

```
        STI     R0,@M1+1        ;Set Machine 1 busy
        LDI     2,R1            ;Request robot, Load part from queue to machine 1
        CALL    _rerobot        ;R0 = initial position, R1 = destination position
                                ;Travel time return in R0
        LDF     @Time,R1
        STF     R1,@CurPart+3   ;Current part entering time
        ADDF3   R0,R1,R2
        STF     R2,@CurPart+4   ;Current process starting time
        LDF     @M1+3,R0
        CALL    _expon          ;Processing time for the part
        ADDF    @CurPart+4,R0
        STF     R0,@EveList2+3  ;Schedule a Part 1 departure after processing
                                ;considering the travel time of robot
        STF     R0,@EveList3+3  ;sending
        STF     R0,@CurPart+5   ;Current part finishing time
        ADDF    @SumM1,R0
        SUBF    @CurPart+4,R0
        STF     R0,@SumM1       ;Update Machine busy period
        B       LA4
*** B   LA4 ;BRANCH OCCURS
;If machine is not available (either busy, blocked or broken), put the current part in queue
LA1     LDI     @NumInQ,R1
        ADDI    1,R1
        STI     R1,@NumInQ      ;Increase the length of queue
        CMPI    20,R1
        BLE     LA2
        LDI     2,R0            ;Queue is overflow, exit 2
        CALL    _exit
LA2     LDI     @NumInQ,AR0     ;InputQ[NumInQ] = InputQ[0]
        MPYI    6,AR0
        ADDI    @PInputQ,AR0
        LDI     @PInputQ,AR1
        LDI     5,RC
        RPTB    LA3
        LDI     *AR1++,R0       ;
LA3     STI     R0,*AR0++
;Schedule next Part A arrival
LA4     LDF     @MeArriA,R0
        CALL    _expon
        ADDF    @Time,R0
        STF     R0,@InputQ0+3
        STF     R0,@EveList1+3  ;Next arrival time
        LDI     @TotNum,R1      ;Update total entering part number
        ADDI    1,R1
        STI     R1,@TotNum
        RETS

*---------------------------------------------------------------------------
* FUNCTION DEF : _depart
* handle current departure event
* Input        : None
* Register     : R0, R1, R2, R3, R4, R5, AR0, AR1, RC, RS, RE, DP
* Output       : None
*---------------------------------------------------------------------------
_depart:
        LDI     @M1+1,R1        ;
        CMPI    3,R1            ;Is Lathe 1 broken?
        BNZ     LD2
        LDF     @EveList4+3,R1
        STF     R1,@CurPart+5   ;Update the finished time for current part
        STF     R1,@EveList2+3  ;Postponed current departure until the machine is up
        B       LDE

LD2     LDF     @SumStay,R2     ;Update part A stay in system
        ADDF    @CurPart+5,R2
        SUBF    @CurPart+3,R2
        STF     R2,@SumStay

;Schedule next part departure
        LDI     @NumInQ ,R3     ;Is the input queue empty?
        BNZ     LD4             ;No, go to LD4
```

```
        LDI     0,R1
        STI     R1,@M1+1        ;Yes, set machine 1 idle, R1=0
        LDF     @BigNum+1,R3
        STF     R3,@EveList2+3  ;No next departure
        STF     R3,@EveList3+3  ;No sending event
        B       LDE
*** B   LD5  ;BRANCH OCCURS
LD4     LDI     1, R0
        STI     R0,@M1+1        ;Machine busy R0 = 1
        LDI     @NumInQ,R3      ;Reduce queue length
        SUBI    R0,R3
        STI     R3,@NumInQ
        LDF     @Time,R5        ;Update delay accumulator
        SUBF    @InputQ0+9,R5
        LDF     @SumDelay,R3
        ADDF    R5,R3
        STF     R3,@SumDelay

        LDI     @PInputQ,AR0
        ADDI    6,AR0
        LDI     @PCurPart,AR1
        LDI     5,RC
        RPTB    LD6
        LDI     *AR0++,R0       ;CurPart = InputQ0
LD6     STI     R0,*AR1++

;Maintain the queue
        LDI     @PInputQ,AR0
        ADDI    6,AR0           ;First in queue
        LDI     6,R0            ;
        ADDI3   R0,AR0,AR1      ;Next in queue
        LDI     @NumInQ,R0
        ADDI    1,R0            ;Fill the last position with zeros
        MPYI    6,R0            ;The total number need to be relocated
        SUBI    1,R0            ;
        LDI     R0,RC           ;Repeat (NumInQ+1)*6 times
        RPTB    LD7
        LDI     *AR1++,R1
LD7     STI     R1,*AR0++

;Schedule next departure and sending
        LDI     2,R0
        LDI     3,R1
        CALL    _rerobot        ;Request robot
        ADDF    @Time,R0
        STF     R0,@CurPart+4
        LDF     @M1+3,R0
        CALL    _expon          ;Processing time
        ADDF    @CurPart+4,R0
        STF     R0,@EveList2+3  ;Next departure time
        STF     R0,@EveList3+3  ;Next sending
        STF     R0,@CurPart+5   ;Part finished time
        ADDF    @SumM1,R0       ;Update Machine busy period
        SUBF    @CurPart+4,R0
        STF     R0,@SumM1
LDE     RETS


*---------------------------------------------------------------------------
* FUNCTION DEF : _send
* handle route part A from Lathe 1 to Mill 1
* Input        : None
* Register     : R0, R1, R2, R3,AR0, AR1
* Output       : None
*---------------------------------------------------------------------------
_send:

LS0     LDI     *AR6,R0         ;Check M2Finish to see if previous departure has been processed
or not?
        CMPI    1,R0
        BNZ     LS0             ;If not, wait until microprocessor 2 processed previous departure
```

```
        LDI     @M1+1,R1        ;
        CMPI    3,R1            ;Is Lathe 1 broken?
        BNZ     LS1
        LDF     @EveList4+3,R1
        STF     R1,@EveList3+3  ;Postponed the current sending event
        B       LSE
LS1     LDI     *+AR6(1),R0
        CMPI    1,R0            ;Check to see if Mill 1 WIP queue is full
        BNZ     LS15
        LDF     *+AR6(4),R1
        CMPF    @EveList3+3,R1  ;If Departure time < M2D_Time?
        BLE     LSE             ;No, wait until M2 process the next departure
        LDF     R1,R5           ;Yes, save the possible departure time
        SUBF    @Time,R5        ;Blocked time period
        LDF     @SumBlock,R0
        ADDF    R5,R0
        STF     R0,@SumBlock    ;Update the total block accumulator
        STF     R1,@CurPart+5   ;Update the finished time for current part
;Reschedule the current departure to the new possible departure time
        STF     R1,@EveList2+3
        STF     R1,@EveList3+3
        LDI     2,R1            ;Machine 2 WIP queue full, M1 blocked
        STI     R1,@M1+1
        B       LSE
;Schedule part A arrival on Mill 1
LS15    LDI     @EveList3,R0    ;M1_M2Dev = EveList3
        LDI     @EveList3+1,R1
        LDI     @EveList3+2,R2
        LDF     @EveList3+3,R3
        LDI     @PM1_M2Dev,AR0  ;Point to the shared memory
        STI     R0,*AR0
        STI     R1,*+AR0(1)
        STI     R2,*+AR0(2)
        STF     R3,*+AR0(3)
        LDI     0,R1
        STI     R1,*AR6         ;Event not processed by microprocessor 2
;Machine is up. Schedule a Part A leave from M1 to M2.
;Wait until microprocessor 2 read the previous messages

LS2     LDI     *+AR6(2),R0
        CMPI    1,R0            ;Wait until microprocessor 2 read previous message, check PartArr
flag
        BZ      LS2
        LDI     @PCurPart,AR0  ;M1_M2Par = CurPart
        LDI     @PM1_M2Part,AR1 ;Point to the shared memory
        LDI     5,RC
        RPTB    LS3
        LDI     *AR0++,R0       ;SendPart = CurPart
LS3     STI     R0,*AR1++
        LDI     *-AR0(1),R0
        STI     R0,*-AR1(3)     ;Reset the starting time on Mill 1
        LDI     1,R0
        STI     R0,*+AR6(2)     ;Inform machine 2 part A arrival

LS4     LDF     @BigNum+1,R0
        STF     R0,@EveList3+3  ;Reset sending event
        LDI     @SumPartA,R3    ;Part A throughput
        ADDI    1,R3
        STI     R3,@SumPartA
LSE     RETS

*---------------------------------------------------------------------------
* FUNCTION DEF : _down
* handle machine break down and repair on M1
* Input        : None
* Register     : R0, R1, R2, AR0, AR1
* Output       : None
*---------------------------------------------------------------------------
_down:
        LDI     @M1+1,R0
```

```
        CMPI    3,R0            ;Is Lathe 1 current broken?
        BNZ     M1LO1           ;
        LDI     @M1PState,R0    ;Machine up, restore machine previous state
        STI     R0,@M1+1
        LDF     @Time,R1        ;Working period
        ADDF    @M1+5,R1
        STF     R1,@EveList4+3  ;Schedule next break down time
        LDI     0,R0
        STI     R0,*+AR5(3)     ;Set no more part A will be routed
        STI     R0,*AR5         ;Inform M2 that Lathe 1 is not down
        LDF     @BigNum+1,R1
        STF     R1,@EveList5+3  ;Set next routing event to be infinite
        B       M1LOE
M1LO1   STI     R0,@M1PState    ;Save current machine status
        LDI     3,R1
        STI     R1,@M1+1        ;Set machine down
        LDF     @Time,R2        ;Repair period
        ADDF    @M1+6,R2
        STF     R2,@EveList4+3  ;Schedule next machine up
        LDI     1,R0
        STI     R0,*+AR5(3)     ;Set routing request to Lathe 2
        STI     R0,*AR5         ;Inform M2 that Lathe 1 is down

M1LOE   RETS

*--------------------------------------------------------------------------
* FUNCTION DEF : _route
* handle when machine break down route part A from Lathe 1 to Lathe 2
* Input        : None
* Register     : R0, R1, R2, R3,R4,R5,AR0, AR1
* Output       : None
*--------------------------------------------------------------------------
_route:
        LDI     @NumInQ,R0      ;Any part in Lathe 1 queue?
        BNZ     M1RO1           ;There is part to route right now
        LDI     2,R0
        STI     R0,*+AR5(1)     ;Inform M2 there is no part to route currently
        B       M1RO3

M1RO1   LDI     *+AR5(1),R0     ;
        CMPI    1,R0
        BZ      M1RO1           ;Wait until last routed part read by M2
        LDI     @PInputQ,AR0    ;Point to the Lathe 1 input queue
        ADDI    6,AR0           ;The first part in queue
        LDI     *AR0,R0         ;Get the first part
        LDI     *+AR0(1),R1
        LDI     *+AR0(2),R2
        LDF     *+AR0(3),R3
        LDF     *+AR0(4),R4
        LDF     *+AR0(5),R5
        LDI     @PM1_M3Part,AR0 ;Point to the shared memory storage
        STI     R0,*AR0         ;Send the first part in queue
        STI     R1,*+AR0(1)
        STI     R2,*+AR0(2)
        STF     R3,*+AR0(3)
        STF     R4,*+AR0(4)
        STF     R5,*+AR0(5)
        LDI     1,R0
        STI     R0,*+AR5(1)     ;Set routed part arrive M2
;Maintain the queue
        LDI     @PInputQ,AR0
        ADDI    6,AR0           ;First in queue, AR0
        LDI     6,R0            ;
        ADDI3   R0,AR0,AR1      ;Next in queue, AR1
        LDI     @NumInQ,R0
        MPYI    6,R0            ;The total number need to be relocated
        SUBI    1,R0            ;
        LDI     R0,RC           ;Repeat (NumInQ+1)*6 times
        RPTB    M1RO2
        LDI     *AR1++,R1
M1RO2   STI     R1,*AR0++       ;Move one part up
```

```
        LDI     @NumInQ,R0      ;Reduce queue length
        SUBI    1,R0
        STI     R0,@NumInQ

M1RO3   LDF     @BigNum+1,R1    ;Working period
        STF     R1,@EveList5+3  ;Schedule next routing event time
M1ROE   RETS

*---------------------------------------------------------------------------
* FUNCTION DEF : _rerobot
* handle robot request, return robot travel time.
* Input        : R0,initial position, R1, destination position
* Register     : R0, R1, AR0, AR1, IR0, IR1, (only AR0,AR1, IR0,IR1 restored)
* Output       : R0 travel time
*---------------------------------------------------------------------------
_rerobot:
;Calculate travel time, robot need to travel from current position to initial position,
;then from initial position to the destination
        PUSH    AR0
        PUSH    AR1
        PUSH    IR0
        PUSH    IR1

        LDI     @PGrephon, AR1          ;Pointer to the robot
        LDI     *+AR1(1),IR0
        MPYI    8,IR0
        ADDI    R0,IR0                  ;From current position to starting position
        LDI     @PRNetwork,AR0
        LDI     R0,IR1
        MPYI    8,IR1
        ADDI    R1,IR1                  ;From starting position to destination
;Robot traveltime = RNetwork[Grephon.R_Pos][initpos] + RNetwork[initpos][destpos]
        ADDF3   *+AR0(IR1),*+AR0(IR0),R0
        STI     R1,*+AR1(1)             ;Next robot position
        LDF     @Time,R1
        CMPF    *+AR1(2),R1
        BGE     LR1
        ADDF    *+AR1(2),R0             ;Robot is occupied, considering some waiting time
        SUBF    R1,R0
;Next time robot stay at initial position and available time is current time plus travel time
LR1     ADDF    R0,R1                   ;Next available time
        STF     R1,*+AR1(2)
        LDF     2.0,R1
        MPYF    R0,R1
        ADDF    *+AR1(4),R1             ;Update robot utilization
        STF     R1,*+AR1(4)

        POP     IR1
        POP     IR0
        POP     AR1
        POP     AR0

        RETS

*---------------------------------------------------------------------------
* FUNCTION DEF : _report
* report simulation results
* Input        : None
* Register     : R0, R2, R5, AR0
* Output       : None
*---------------------------------------------------------------------------
_report:
        LDF     @EndTime,R0
        CALL    INV_F30
        RND     R0,R5           ;R5=1/Time

        LDF     @SumM1,R0
        MPYF    R5,R0
        STF     R0,@AveM1       ;Utilization of M1

        LDI     @PGrephon,AR0
```

```
        LDF     *+AR0(4),R0
        MPYF    R5,R0
        STF     R0,@AveR        ;Utilization of robot

        LDF     @SumBlock,R0
        MPYF    R5,R0
        STF     R0,@AveBlock    ;Average blocked

        LDF     @SumInQ,R0
        MPYF    R5,R0
        STF     R0,@AveInQ      ;Average waiting in queue

        FLOAT   @SumPartA,R0
        CALL    INV_F30
        RND     R0,R2           ;R2 = 1/SumPart
        LDF     @SumDelay,R0
        MPYF    R2,R0
        STF     R0,@AveDelay    ;Average delay

        LDF     @SumStay,R0
        MPYF    R2,R0
        STF     R0,@AveStay     ;Average stay in FMS
        RETS

*---------------------------------------------------------------------
* FUNCTION DEF : _exit
* Unusually exit
* Input        : none
* Register     : none
* Output       : none
*---------------------------------------------------------------------
_exit:
        LDP     @_main
        STI     R0,@SysExit
EXITSIM B       $

*---------------------------------------------------------------------
* FUNCTION DEF : _barrier
* Unusually exit
* Input        : none
* Register     : R0
* Output       : none
*---------------------------------------------------------------------
_barrier:
        LDP     SharedMem1
        LDI     @Barrier1,R0
        ADDI    1,R0
        STI     R0,@Barrier1
B1      LDI     @Barrier1,R0
        CMPI    2,R0
        BLT     B1
        LDP     SharedMem2
        LDI     @Barrier2,R0
        ADDI    1,R0
        STI     R0,@Barrier2
B2      LDI     @Barrier2,R0
        CMPI    2,R0
        BLT     B2
        LDP     @_main
        RETS

*---------------------------------------------------------------------
* FUNCTION DEF : _expon
* Generate a random number with exponential distribution
* Input        : R0 is the mean
* Register     : R1
* Output       : R0 is the random number with exponential distribution
*---------------------------------------------------------------------
_expon:
*Fast 32 bit uniform random number generator
        PUSH    R1
```

```
        PUSHF   R1

        NEGF    R0, R1          ; R1 is the negative of the mean
        PUSH    R1
        PUSHF   R1              ; Save -mean in the stack, all 40 bits
        LDI     @SEED,R0        ; Call here for last SEED
        MPYI    @A,R0           ; Calculate r = (A*r+C)%m
        ADDI    @C,R0           ;
        AND     @MASK1, R0      ; m=2^23
        STI     R0,@SEED        ; Result is returned
        ADDI    1,R0            ;
        FLOAT   R0,R0           ;
        MPYF    @INVE23,R0      ; r/m
        CALL    _ln             ; ln(u)
        POPF    R1              ; -Mean
        POP     R1
        MPYF    R1, R0          ; random number =  -mean*ln(u)

        POPF    R1
        POP     R1
        RETS


*-------------------------------------------------------------------------
*Define references
*-------------------------------------------------------------------------
*Define the entry point of subroutines
SUB     .word _main
        .word _arrive
        .word _depart
        .word _send
        .word _down
        .word _route
PSUB    .word SUB

        .sect "isrcode"
*Interrupt Service Routine
_ISRCount:
        PUSH DP
        PUSH R0
        PUSHF R0
        LDP _main
        LDI @COUNT, R0
        ADDI 1, R0
        STI R0, @COUNT
        POPF R0
        POP R0
        POP DP
        RETI

STACK   .word  $+1           ; Bottom of the Stack
        .end
```

C.3.2.2 Code running on DSP 2

```
*-------------------------------------------------------------------
* 4MHardM1.ASM -- conservative parallel discrete event simulation
* of a scenario: 4 machine, 2 part type, 2 robot, and considering machine breakdown
* This is the program running on Microprocessor 2, using to simulating Mill 1
* Programmed by: Dong Xu
* Date         : Mar.22, 2001
* Modified     : Apr.3, 2001
*-------------------------------------------------------------------
        .start "data", 0x809900    ;Data section, 809900-809AFF
        .start "code", 0x809B00     ;Code section, 809B00-809EFF
        .start "shared1", 0xA00000   ;Shared memory section, A00000-A003FF
        .start "text", 0xB00000    ;Main text section, B00000-B007FFF

        .sect "data"
;These global variables are used to pass parameters between functions to execute the simulation
;Average of simulation results
SysExit     .int 5      ;Exit state of the simulation
;                            0 success, 1 event list empty, 2 Lathe 1 input queue overflow
;                            3 Mill 1 input queue overflow, 5 other fatal error, 6 ln negative
AveInQ      .float 0.0  ;Average of number parts in queue for part A
            .float 0.0  ;Part B
AveM2       .float 0.0  ;Average of machine 2 utilization
AveR        .float 0.0  ;Average of robot utilization
AveStay     .float 0.0  ;Average time stay in system
            .float 0.0
AveDelay    .float 0.0  ;Average delay time of parts
            .float 0.0

NumInQ      .int 0      ;Number of parts waiting in M2 input queue
            .int 0
TotNum      .int 0      ;Total number of parts entering
            .int 1
SumPart     .int 0      ;Total finished Parts
            .int 0

SumInQ      .float 0.0  ;Sum of number parts in queues
            .float 0.0
SumM2       .float 0.0  ;Sum of machine 2 utilization
SumDelay    .float 0.0  ;Total delay time of parts
            .float 0.0
SumStay     .float 0.0  ;Total time stay in system
            .float 0.0
EndTime     .float 14400.0;End simulation time, 240 hours
MeArriB     .float 20.0 ;Mean time between arrival of Part B

Time        .float 0.0  ;Current simulating time, in minutes
LastTime    .float 0.0  ;Time of last event
NxtEvet     .int 0      ;Next event type, 0=empty, 1=part B arrival, 2=part A arrival
                        ;3=parts departure
;Event list
;current event
EveList0    .int 0      ;Event type = 0(current), 1(Arrival1), 2(Arrival2), 3(Departure)
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 2      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
;Part B arrival event
EveList1    .int 1      ;Event type = 0(current), 1(Arrival1), 2(Arrival2), 3(Departure)
            .int 1      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 2      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
;Part A arrival event
EveList2    .int 2      ;Event type = 0(current), 1(Arrival1), 2(Arrival2), 3(Departure))
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 2      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
;departure event
EveList3    .int 3      ;Event type = 0(current), 1(Arrival1), 2(Arrival2), 3(Departure)
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 2      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
```

```
WIPQ0       .space 42   ;Working-in-process queue for part A, = 6*6+6 = 42
InputQ0     .space 132  ;Input queue for part B, =6*21+6 for tail

;Defined the structure used in the program
;Describe workstation 2
M2          .int 2      ;Machine type, 1 = lathe, 2 = mill
            .int 0      ;Machine state = 0(idle), 1(busy), 2(blocked), 3(broken)
            .int 0      ;Current operating part type, 0 = Part A, 1 = Part B
            .float 8.0  ;Processing time for Part A
            .float 4.0  ;Processing time for Part B, not available
            .float 0.0  ;Mean time between failure
            .float 0.0  ;Mean time to repair

;Current part scheduled to be processed on machine
CurPart     .int 1      ;Part type = 0(Part A), 1(Part B)
            .int 0      ;Operation required on workstation 1, 0(No), 1(Yes)
            .int 1      ;Operation required on workstation 2, 0(No), 1(Yes)
            .float 0.0  ;Time when entering the FMS
            .float 0.0  ;Processing start time to one workstation
            .float 0.0  ;Finish time of one operation on one workstation

;Define the pointers to the arrays
PAveInQ     .word AveInQ
PAveDelay   .word AveDelay
PAveStay    .word AveStay
PNumInQ     .word NumInQ
PTotNum     .word TotNum
PSumInQ     .word SumInQ
PSumPart    .word SumPart
PSumDelay   .word SumDelay
PSumStay    .word SumStay
PM2         .word M2
PCurPart    .word CurPart
PEveList    .word EveList0
PInputQ     .word InputQ0
PWIPQ       .word WIPQ0
PRNetwork   .word RNetwork1
PM1_M2Part  .word M1_M2Part
PM1_M2Dev   .word M1_M2Dev
PGrephon    .word Grephon
PSharedMem1 .word M2Finish

*-------------------------------------------------------------------------
* DEFINE CONSTANTS
*-------------------------------------------------------------------------
QLimit      .int 20             ;Length limit of the queue
WIPLimit    .int 5              ;Limit of WIP queue
NumEvet     .int 3              ;Number of event
NumPType    .int 2              ;Number of part type

;Constant used by random number generation subroutine
A       .int    1078373         ; Constants needed for RAND
C       .int    2311527         ;
SEED    .int    0       ;
MASK    .word   0xFF7FFFFF      ;Mask for fast inverse float
MASK1   .word   0x007FFFFF      ;Mask for exponential distribution, %2^23
INVE23  .word   0xE9000000      ;1/2^23
BigNum  .float  1.0e29          ;Constant big value
        .float  1.0e30          ;

;Internal constants for ln(u)
;Scaling coefficients for ln(1+x)
LNRM    .float  0.6931471806    ; LN(2)
C0      .float  1.0000000000    ; C0 (1.0)
;       Polynomial coefficients for ln(1+X), 0 <= X < 1.
        .float  0.9999964239    ; TOP OF C1
        .float  -0.4998741238   ; TOP OF C2
        .float  0.3317990258    ; TOP OF C3
        .float  -0.2407338084   ; TOP OF C4
        .float  0.1676540711    ; TOP OF C5
```

```
        .float  -0.0953293897    ; TOP OF C6
        .float  0.0360884937     ; TOP OF C7
C8      .float  -0.0064535442    ; TOP OF C8
AC8     .word   C8

*-------------------------------------------------------------------------------
*Special signals for synchronization purpose, should be stored in the shared memory,
*accessed by to both Microprocessor
*-------------------------------------------------------------------------------
    .sect "shared1"
M2Finish    .int 1      ;Indicate M2 finish the previous part departure event, *AR6
M2QFull     .int 0      ;Signal of WIP queue, 0 = not full, 1 = full, *+AR6(1)
PartArr     .int 0      ;Indicate Part A leave from Machine 1 to 2, *+AR6(2)
M2SafeTime  .float 0.0  ;Indicate how far simulation clock on M2 can advance, *+AR6(3)
M2D_Time    .float 0.0  ;Scheduled next part departure time on machine 2, *+AR6(4)

;Departure message from Machine 1 to Machine
M1_M2Dev    .int 0      ;Event type = 0(current), 1(Arrival), 2(Departure), 3(Machine Broken)
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 1      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
;Current Part transported from Machine 1 to Machine 2
M1_M2Part   .int 0      ;Part type = 0(Part A), 1(Part B)
            .int 1      ;Operation required on workstation 1, 0(No), 1(Yes)
            .int 1      ;Operation required on workstation 2, 0(No), 1(Yes)
            .float 0.0  ;Time when entering the FMS
            .float 0.0  ;Arrival time to one workstation
            .float 0.0  ;Finish time of one operation on one workstation

;Describe the robot
Grephon     .int 0      ;Robot state = 0(idle), 1(occupied)
            .int 1      ;current position of robot
            .float 0.0  ;Next available time of the robot
            .float 0.0  ;Next available time of the robot
            .float 0.0  ;Area of robot utilization

;Robot 1 path network, using real travel time if necessary
RNetwork1   .float 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
            .float 0.0, 0.0, 0.3, 0.6, 0.9, 1.2, 1.5, 0.3
            .float 0.0, 0.3, 0.0, 0.3, 0.6, 0.9, 1.2, 0.3
            .float 0.0, 0.6, 0.3, 0.0, 0.3, 0.6, 0.9, 0.6
            .float 0.0, 0.9, 0.6, 0.3, 0.0, 0.3, 0.6, 0.6
            .float 0.0, 1.2, 0.9, 0.6, 0.3, 0.0, 0.3, 0.9
            .float 0.0, 1.5, 1.2, 0.9, 0.6, 0.3, 0.0, 0.9
            .float 0.0, 0.3, 0.3, 0.6, 0.6, 0.9, 0.9, 0.0

SharedMem1  .word M2Finish
Barrier1    .int 0

    .sect "code"
    .entry  _main
*-------------------------------------------------------------------------------
* FUNCTION DEF : _main
* Control the advance of the program
*-------------------------------------------------------------------------------
_main:
        LDP     @_main          ;Set Data Segment Pointer
        LDI     @STACK,SP       ;Set stack pointer
        LDI     @PSharedMem1,AR6

        CALL    _initial        ;Initialize the simulation, especially data structure
        CALL    _barrier        ;barrier for starting

LM1     LDF     @Time,R0
        CMPF    @EndTime,R0     ;Does simulation end?
        BGT     LM2

        CALL    _timing         ;Determine the next event type

;Invoke the appropriate event handle function
        LDI     @PSUB,AR0
```

```
        ADDI    @NxtEvet,AR0
        LDI     *AR0,R0
        CALLU   R0              ;Handle parts arrival, departure
        B       LM1
LM2
        CALL    _report         ;Report simulation results

        LDI     0,R0
        STI     R0,@SysExit     ;Successfully run


END     B       $               ;End of the program


*-------------------------------------------------------------------------
* FUNCTION DEF : _initial
* Fill the robot path network, initialize the data structures and the first arrival
* Input        : None
* Register     : R1, AR0, RC, RS, RE
* Output       : None
*-------------------------------------------------------------------------
_initial:
;Describe the current part on workstation 2, WIPQ0[0] is the storage of next part A
        LDI     0,R1
        STI     R1,@WIPQ0       ;Part A
        LDI     1,R1
        STI     R1,@WIPQ0+1     ;Part A need turning operation on workstation 1
        STI     R1,@WIPQ0+2     ;Part A need drilling operation on workstation 2
        LDF     0.0,R1
        STF     R1,@WIPQ0+3     ;Part A first arrival time
        STF     R1,@WIPQ0+4     ;Part A start on workstation 2
        STF     R1,@WIPQ0+5     ;Time to finish operation on workstation 2
;Describe the part B on workstation 2, InputQ0[0] is the storage of next part
        LDI     1,R0
        STI     R0,@InputQ0     ;Part B
        LDI     0,R1
        STI     R1,@InputQ0+1   ;Part B do not need turning operation on workstation 1
        STI     R0,@InputQ0+2   ;Part B need drilling operation on workstation 2
        LDF     @MeArriB,R0
        CALL    _expon
        STF     R0,@InputQ0+3   ;Part B first arrival time
        LDF     0.0,R1
        STF     R1,@InputQ0+4   ;Part B start on workstation 2, depending on robot
        STF     R1,@InputQ0+5   ;Time to finish operation on workstation 2 does not known yet
;Schedule first arrivals
        LDF     @InputQ0+3,R1
        STF     R1,@EveList1+3  ;Schedule the part B arrive time
        LDF     @BigNum+1,R1
        STF     R1,@EveList2+3  ;The arrival time of part A is passed by Microprocessor 1
        STF     R1,@EveList3+3  ;Schedule the initial departure time to be infinite
        RETS

*-------------------------------------------------------------------------
* FUNCTION DEF : _timing
* schedule the next event, synchronize and advance the simulation clock
* Input        : None
* Register     : R0, R1, R2, R3, R4, R5, AR0, AR1, DP, RC, RS, RE
* Output       : None
*-------------------------------------------------------------------------
_timing:
        NOP
LT1     LDI     *+AR6(2),R0     ;Check for PartArr flag
        CMPI    1,R0            ;Synchronize with microprocessor 1, check for new part A arrival
        BNZ     LT15
;Schedule a part A arrival
        LDI     @PM1_M2Dev,AR0  ;Point to M1_M2Dev
        LDI     *AR0,R0         ;Event List 2 = M1_M2Dev
        LDI     *+AR0(1),R1
        LDI     *+AR0(2),R2
        LDF     *+AR0(3),R3
        STI     R0,@EveList2
```

```
        STI     R1,@EveList2+1
        STI     R2,@EveList2+2
        STF     R3,@EveList2+3

;Determine the minimum time among event list and set the next event type
;If two event happen at the same time, handle machine broken event first, then
;departure event, and the last is arrival event
LT15    LDF     @BigNum,R1      ;A initial min time
        LDI     0,R0
        STI     R0,@NxtEvet     ;Next event type = 0
        LDI     1,R0            ;Iteration index
        LDI     @PEveList,AR0
        ADDI    3,AR0           ;Point to the first Event List E_Time
LT2     CMPF    *++AR0(4),R1    ;Event List[i].E_Time <= min_time?
        BLT     LT3
        LDF     *AR0,R1         ;Minimum time is
        STI     R0,@NxtEvet     ;Next Event No.
LT3     ADDI    1,R0
        CMPI    3,R0            ;Need to compare all 3 events
        BLE     LT2

        LDI     @NxtEvet,R0     ;If event list is empty
        BNZ     LT4
        LDI     1,R0            ;Exit(1), event list is empty
        CALL    _exit           ;If even list is empty, stop the simulation and exit
LT4     CMPF    @EndTime, R1
        BGT     LT5             ;If Time > simulation end time, then jump out
        CMPF    *+AR6(3),R1     ;Check if can safely process the next event
        BGT     LT1             ;Workstation 2 cannot process event has time stamp
                                ;larger than the SafeTime.
                                ;SafeTime must be provided by microprocessor 1
LT5     STF     R1,@Time        ;Simulation clock
        LDF     @LastTime,R0    ;Last event time
        SUBF3   R0,R1,R2        ;Compute time since last event
        STF     R1, @LastTime   ;Save last event time

        LDI     @PNumInQ,AR0    ;Pointer of NumInQ
        LDI     @PSumInQ,AR1    ;Pointer SumInQ
        LDI     1,RC            ;Repeat 2 times
        RPTB    LT6
        FLOAT   *AR0++,R0
        MPYF    R2,R0
        ADDF    *AR1,R0
LT6     STF     R0,*AR1++       ;Update area of number in queues
LTE     RETS

*---------------------------------------------------------------------------
* FUNCTION DEF : _arriveb
* schedule the next arrival event and handle current arrival event
* Input        : None
* Register     : R0, R1, R2, R4, AR0, AR1, RC, RS, RE
* Output       : None
*---------------------------------------------------------------------------
_arriveb:
;Check to see if Mill 1 is idle
        LDI     @M2+1,R0
        BNZ     LA4             ;M2 is not idle

        LDI     1,R0            ;M2 is idle, set it to busy
        STI     R0,@M2+1
;Load part from queue to machine 2, WIP to M2 is 3->5, Input Queue to M2 is 4->5,
        LDI     5,R1            ;Destination
        LDI     4,R0            ;Starting
        CALL    _rerobot        ;Travel time return in R0
        LDF     @Time,R1
        STF     R1,@CurPart+3   ;Entering time
        ADDF    R0,R1
        STF     R1,@CurPart+4   ;Start processing time
        LDI     @PM2,AR0
        LDF     *+AR0(4),R0     ;M2.P_Time[PartB]
        CALL    _expon
```

```
;Schedule a Part departure after processing, considering the travel time
        ADDF    @CurPart+4,R0   ;Considering the travel time of robot
        STF     R0,@EveList3+3  ;Next departure time
        STF     R0,@CurPart+5   ;Current part finished time
        LDI     1,R4            ;Part type B = 1
        STI     R4,@EveList3+1  ;Next departure part type
        STI     R4,@CurPart
;Update Machine 2 busy period
        ADDF    @SumM2,R0
        SUBF    @CurPart+4,R0
        STF     R0,@SumM2
        B       LA8
*** B   LA8 ;BRANCH OCCURS
LA4     LDI     @PNumInQ,AR0    ;M2 is not idle
        LDI     *+AR0(1),R1
        ADDI    1,R1
        STI     R1,*+AR0(1)     ;Increase queue length
        LDI     @QLimit,R2      ;Check if input queue full
        CMPI    *+AR0(1),R2
        BGE     LA6
        LDI     3,R0
        CALL    _exit
LA6     LDI     @PInputQ,AR1    ;The first position of the queue
        LDI     *+AR0(1),R1     ;number in queues
        MPYI    6,R1
        ADDI    R1,AR1          ;InputQ[PartB][NumInQ[PartB]]
        LDI     @PInputQ,AR0
        LDI     5,RC            ;6 data to pass
        RPTB    LA7
        LDI     *AR0++,R1
LA7     STI     R1,*AR1++       ;InputQ[PartType][NumInQ[PartType]] = InputQ[PartType][0]
;If Part B arrive, schedule next Part B arrival
LA8     LDF     @MeArriB,R0     ;Schedule next Part B arrival
        CALL    _expon
        ADDF    @Time,R0
        STF     R0,@InputQ0+3
        STF     R0,@EveList1+3

        LDI     @PTotNum,AR0    ;Increase total entering part number
        LDI     1,R0
        ADDI    *+AR0(1),R0
        STI     R0,*+AR0(1)
        RETS

*---------------------------------------------------------------------------
* FUNCTION DEF : _arrivea
* schedule the next arrival event and handle current arrival event
* Input        : None
* Register     : R0, R1, R2, R4, AR0, AR1, RC, RS, RE
* Output       : None
*---------------------------------------------------------------------------
_arrivea:
LAA0    LDI     *+AR6(2),R0
        CMPI    1,R0
        BNZ     LAA0            ;Check if new part arrive
;Read in arrived part
        LDI     @PM1_M2Part,AR0 ;Point to shared memory
        LDI     *AR0,R0         ;WIP queue 0 = M1_M2Par
        LDI     *+AR0(1),R1
        LDI     *+AR0(2),R2
        LDF     *+AR0(3),R3
        LDF     *+AR0(4),R4
        LDF     *+AR0(5),R5
        STI     R0,@WIPQ0
        STI     R1,@WIPQ0+1
        STI     R2,@WIPQ0+2
        STF     R3,@WIPQ0+3
        STF     R4,@WIPQ0+4
        STF     R5,@WIPQ0+5
        LDI     0,R0
        STI     R0,*+AR6(2)     ;Parts read
```

```
;Check to see if Mill 1 is idle
        LDI     @M2+1,R0
        BNZ     LB4             ;M2 is not idle

        LDI     1,R0            ;M2 is idle, set it to busy
        STI     R0,@M2+1
;Load part from queue to machine 2, WIP to M2 is 3->5, Input Queue to M2 is 4->5,
        LDI     5,R1            ;Destination
        LDI     3,R0            ;Starting
        CALL    _rerobot        ;Travel time return in R0
        LDF     @Time,R1
        STF     R1,@CurPart+3   ;Entering time
        ADDF    R0,R1
        STF     R1,@CurPart+4   ;Start processing time
        LDI     @PM2,AR0
        LDF     *+AR0(3),R0     ;M2.P_Time[PartA]
        CALL    _expon
;Schedule a Part departure after processing, considering the travel time
        ADDF    @CurPart+4,R0   ;Considering the travel time of robot
        STF     R0,@EveList3+3  ;Next departure time
        STF     R0,@CurPart+5   ;Current part finished time
        LDI     0,R4            ;Part type A = 0
        STI     R4,@EveList3+1  ;Next departure part type
        STI     R4,@CurPart
;Update Machine 2 busy period
        ADDF    @SumM2,R0
        SUBF    @CurPart+4,R0
        STF     R0,@SumM2
        B       LB8
*** B   LA8 ;BRANCH OCCURS
LB4     LDI     @PNumInQ,AR0    ;M2 is not idle
        LDI     *AR0,R1
        ADDI    1,R1
        STI     R1,*AR0         ;Increase queue length
        LDI     @WIPLimit,R2    ;Check if input queue full
        CMPI    *AR0,R2
        BGT     LB6
        LDI     1,R1
        STI     R1,*+AR6(1)     ;WIP queue full
        LDF     @EveList3+3,R0  ;Next Departure time
        STF     R0,*+AR6(4)     ;Inform M1 next part departure time till block released

LB6     LDI     @PWIPQ,AR1      ;The first position of the queue
        LDI     *AR0,R1         ;number in queues
        MPYI    6,R1
        ADDI    R1,AR1          ;InputQ[PartB][NumInQ[PartB]]
        LDI     @PWIPQ,AR0
        LDI     5,RC            ;6 data to pass
        RPTB    LB7
        LDI     *AR0++,R1
LB7     STI     R1,*AR1++       ;InputQ[PartType][NumInQ[PartType]] = InputQ[PartType][0]
;Schedule next Part A arrival to be infinite
LB8     LDI     1,R1
        STI     R1,*AR6         ;Indicate the event has been processed

        LDF     @BigNum+1,R0
        STF     R0,@EveList2+3  ;Reset the part A arrival event
        LDI     @PTotNum,AR0    ;Increase total entering part number
        LDI     1,R0
        ADDI    *AR0,R0
        STI     R0,*AR0
        RETS

*---------------------------------------------------------------------------
* FUNCTION DEF : _depart
* handle current departure event
* Input        : None
* Register     : R0, R1, R2, R3, R4, R5, AR0, AR1, RC, RS, RE, DP
* Output       : None
*---------------------------------------------------------------------------
```

```
_depart:
        LDI     @EveList3+1,R4  ;Current part type in R4
;Update current part stay in system, it is realy partial of part stay, need to plus
;the stay time at machine 1
        LDF     @CurPart+5,R0
        SUBF    @CurPart+3,R0
        LDI     @PSumStay,AR0
        ADDI3   R4,AR0,AR1      ;
        ADDF    *AR1,R0
        STF     R0,*AR1         ;SumStay[PartType] += CurPart.P_Finish - CurPart.P_Enter
        LDI     @PSumPart,AR0   ;Update finished parts
        ADDI3   R4,AR0,AR1
        LDI     1,R0
        ADDI3   R0,*AR1,R1
        STI     R1,*AR1
;Check to see if all queues are empty
        LDI     @NumInQ,R1
        BNZ     LD1             ;WIP queue is not empty
        LDI     @NumInQ+1,R2
        BNZ     LD1             ;Input queue is not empty
        LDI     0,R3            ;All queues are empty
        STI     R3,@M2+1        ;Set Machine 2 idle
        STI     R3,*+AR6(1)     ;M2 queues are not full
        STI     R4,@EveList3+1  ;Part type does not matter
        LDF     @BigNum+1,R3
        STF     R3,@EveList3+3  ;Set next departure event infinite
        B       LDE
*** B   LDE  ;BRANCH OCCURS
;Queues are not empty, decide next part to be processed according to the dispatching rule.
;Here FIFO is adopted
LD1     LDI     @NumInQ,R1
        BNZ     LD2
        LDI     1,R4            ;If WIP queue is empty, next part is B
        B       LD5
LD2     LDI     @NumInQ+1,R2
        BNZ     LD3
        LDI     0,R4            ;If Input queue is empty, next part is A
        B       LD5
LD3     LDF     @WIPQ0+9,R3     ;If both are not empty, FIFO
        CMPF    @InputQ0+9,R3   ;if WIPQ[1].P_Enter <= INPUTQ[1].P_Enter
        BGT     LD4             ;If part B arrival first
        LDI     0,R4            ;Part A arrive first
        B       LD5
LD4     LDI     1,R4            ;Part B arrive first
LD5     LDI     1,R0
        STI     R0,@M2+1        ;Set Machine 2 busy
        LDI     @PNumInQ,AR1
        ADDI    R4,AR1
        SUBI3   R0,*AR1,R2      ;Reduce queue length
        STI     R2,*AR1
;Reset WIP queue is not full
        LDI     @NumInQ,R1      ;M2QFull = (NumInQ[0]<WIPLimit)?0:1
        CMPI    @WIPLimit,R1
        LDILT   0,R2
        LDIGE   1,R2
        STI     R2,*+AR6(1)     ;M2 WIP Queue not full, M2QFULL

;Update delay accumulator
        LDI     42,R3
        MPYI3   R3,R4,AR0       ;Index position
        ADDI    @PWIPQ,AR0      ;Head of queues
        LDI     6,R0
        ADDI3   R0,AR0,AR1      ;First position of queues
        LDI     3,R0
        ADDI3   R0,AR1,AR0      ;First Part in queue Entering time
        LDF     @Time,R5        ;Current simulation clock
        SUBF    *AR0,R5         ;Delay time = Time - P_Enter
        LDI     @PSumDelay,AR0
        ADDI    R4,AR0
        ADDF3   R5,*AR0,R3      ;Sum of delay
        STF     R3,*AR0
```

```
;Current Part
        LDI     @PCurPart,AR0   ;Current part pointer in AR0, AR1 points to the first in queue
        LDI     5,RC
        RPTB    LD6
        LDI     *AR1++,R3
LD6     STI     R3,*AR0++       ;CurPart = InputQ[PartType][1]
;Maintain the queue
        LDI     42,R3
        MPYI3   R3,R4,AR0       ;Index position
        ADDI    @PWIPQ,AR0      ;Head of queues
        LDI     6,R0
        ADDI    R0,AR0          ;First in queues
        ADDI3   R0,AR0,AR1      ;Next in queue

        LDI     @PNumInQ,AR2    ;Point to the NumInQ
        ADDI    R4,AR2
        LDI     *AR2,R0
        ADDI    1,R0            ;Fill the last position with zeros
        MPYI    6,R0            ;The total number need to be relocated
        SUBI    1,R0
        LDI     R0,RC
        RPTB    LD7
        LDI     *AR1++,R2
LD7     STI     R2,*AR0++
;Schedule next departure
        LDI     6,R0
        LDI     5,R1            ;Parts leave machine 2 and exit the FMS
        CALL    _rerobot        ;Request robot
        ADDF    @Time,R0
        STF     R0,@CurPart+4   ;Starting time
        LDI     @PM2,AR0
        ADDI3   R4,AR0,AR1
        LDF     *+AR1(3),R0     ;Processing time
        CALL    _expon
        ADDF    @CurPart+4,R0
        STF     R0,@EveList3+3  ;Schedule next departure
        STI     R4,@EveList3+1  ;Event related part type
        STF     R0,@CurPart+5   ;Current part finished time
        STF     R0,*+AR6(4)     ;Inform M1 next part departure time till block released, M2D_Time
;Update Machine busy period
        ADDF    @SumM2,R0
        SUBF    @CurPart+4,R0
        STF     R0,@SumM2
LDE     RETS

*---------------------------------------------------------------------------
* FUNCTION DEF : _rerobot
* handle robot request, return robot travel time.
* Input        : R0,initial position, R1, destination position
* Register     : R0, R1, AR0, IR0, IR1, (only AR0,IR0,IR1 restored)
* Output       : R0 travel time
*---------------------------------------------------------------------------
_rerobot:
;Calculate travel time, robot need to travel from current position to initial position,
;then from initial position to the destination
        PUSH    AR0
        PUSH    AR1
        PUSH    IR0
        PUSH    IR1

        LDI     @PGrephon, AR1          ;Pointer to the robot
        LDI     *+AR1(1),IR0
        MPYI    8,IR0
        ADDI    R0,IR0                  ;From current position to starting position
        LDI     @PRNetwork,AR0
        LDI     R0,IR1
        MPYI    8,IR1
        ADDI    R1,IR1                  ;From starting position to destination
;Robot traveltime = RNetwork[Grephon.R_Pos][initpos] + RNetwork[initpos][destpos]
        ADDF3   *+AR0(IR1),*+AR0(IR0),R0
        STI     R1,*+AR1(1)             ;Next robot position
```

```
        LDF     @Time,R1
        CMPF    *+AR1(3),R1
        BGE     LR1
        ADDF    *+AR1(3),R0             ;Robot is occupied, considering some waiting time
        SUBF    R1,R0
;Next time robot stay at initial position and available time is current time plus travel time
LR1     ADDF    R0,R1                   ;Next available time
        STF     R1,*+AR1(3)
        LDF     2.0,R1
        MPYF    R0,R1
        ADDF    *+AR1(4),R1             ;Update robot utilization
        STF     R1,*+AR1(4)

        POP     IR1
        POP     IR0
        POP     AR1
        POP     AR0

        RETS
*-------------------------------------------------------------------------
* FUNCTION DEF : _report
* report simulation results
* Input        : None
* Register     : R0
* Output       : None
*-------------------------------------------------------------------------
_report:

        LDF     @EndTime,R0
        CALL    INV_F30
        RND     R0,R5           ;R5=1/Time

        LDF     @SumM2,R0
        MPYF    R5,R0
        STF     R0,@AveM2       ;Utilization of M1

        LDI     @PGrephon,AR0
        LDF     *+AR0(4),R0
        MPYF    R5,R0
        STF     R0,@AveR        ;Utilization of robot

        LDI     0,R4
LRE1    LDI     @PSumInQ,AR0
        ADDI    R4,AR0
        LDF     *AR0,R0
        MPYF    R5,R0
        LDI     @PAveInQ,AR0
        ADDI    R4,AR0
        STF     R0,*AR0         ;Average in queues

        LDI     @PSumPart,AR0
        ADDI    R4,AR0
        FLOAT   *AR0,R0
        CALL    INV_F30
        RND     R0,R2           ;R2 = 1/SumPart[i]

        LDI     @PSumDelay,AR0
        ADDI    R4,AR0
        LDF     *AR0,R0
        MPYF    R2,R0
        LDI     @PAveDelay,AR0
        ADDI    R4,AR0
        STF     R0,*AR0         ;Average delay

        LDI     @PSumStay,AR0
        ADDI    R4,AR0
        LDF     *AR0,R0
        MPYF    R2,R0
        LDI     @PAveStay,AR0
        ADDI    R4,AR0
        STF     R0,*AR0         ;Average stay
```

```
        ADDI    1,R4
        CMPI    2,R4
        BLT     LRE1
        RETS

*---------------------------------------------------------------------
* FUNCTION DEF : _barrier
* Unusually exit
* Input        : none
* Register     : R0
* Output       : none
*---------------------------------------------------------------------
_barrier:
        LDP     SharedMem1
        LDI     @Barrier1,R0
        ADDI    1,R0
        STI     R0,@Barrier1
B1      LDI     @Barrier1,R0
        CMPI    2,R0
        BLT     B1
        LDP     @_main
        RETS

*---------------------------------------------------------------------
* FUNCTION DEF : _exit
* Unusually exit
* Input        : none
* Register     : none
* Output       : none
*---------------------------------------------------------------------
_exit:
        LDP     @_main
        STI     R0,@SysExit
EXITSIM B       $

*---------------------------------------------------------------------
* FUNCTION DEF : _expon
* Generate a random number with exponential distribution
* Input        : R0 is the mean
* Register     : R1
* Output       : R0 is the random number with exponential distribution
*---------------------------------------------------------------------
_expon:
*Fast 32 bit uniform random number generator
        PUSH    R1
        PUSHF   R1

        NEGF    R0, R1          ; R1 is the negative of the mean
        PUSH    R1
        PUSHF   R1              ; Save -mean in the stack, all 40 bits
        LDI     @SEED,R0        ; Call here for last SEED
        MPYI    @A,R0           ; Calculate r = (A*r+C)%m
        ADDI    @C,R0           ;
        AND     @MASK1, R0      ; m=2^23
        STI     R0,@SEED        ; Result is returned
        ADDI    1,R0            ;
        FLOAT   R0,R0           ;
        MPYF    @INVE23,R0      ; r/m
        CALL    _ln             ; ln(u)
        POPF    R1              ; -Mean
        POP     R1
        MPYF    R1, R0          ; random number =  -mean*ln(u)

        POPF    R1
        POP     R1
        RETS


*---------------------------------------------------------------------------
*Define references
```

```
*-------------------------------------------------------------------------
*Define the entry point of subroutines
SUB     .word _main
        .word _arriveb
        .word _arrivea       ;parts arrival invoke same subroutine
        .word _depart
PSUB    .word SUB
STACK   .word  $+1          ; Bottom of the Stack
        .end
```

Note:

1. Subroutines _ln and _INVF30 are the same as in 1MhardL1.asm

C.3.2.3 Code running on DSP 3

```
*-----------------------------------------------------------------------
* 4MHardL2.ASM -- parallel discrete event simulation
* of a scenario: 4 machine, two part types, two robots, and considering machine breakdown
* This is the program running on Microprocessor 3, using to simulating Lathe 2
* Programmed by: Dong Xu
* Date         : March 22, 2001
* Modified     : March 27, 2001
*-----------------------------------------------------------------------
        .start "data", 0x809900     ;Data section, 809900-8099FF
        .start "code", 0x809A00     ;Code section, 809A00-809EFF
        .start "isrcode", 0x809D00  ;Interrupt service routine section
        .start "shared1", 0xA00000  ;Shared memory section 1, A00000-A003FF
        .start "shared2", 0xC00000  ;Shared memory section 2, C00000-C003FF
        .start "text", 0xB00000     ;text section, B00000-B07FFF

        .sect "data"
;Constants for Timer 0 control
EINT    .word 00002000h    ;Enable global interrupt
ETINT0  .word 00000100h    ;Enable Timer0 interrupt
PERIOD  .int 100000        ;Timer period
BISR    .word 60809D00h    ;Branch to interrupt service routine, 60 jump to 0x809D00h
COUNT   .int 0             ;Count of Timer interrupt
TOTCYC  .int 0             ;Total cycles
SysExit .int 5             ;Exit state of the simulation,
;                            0 success, 1 event list empty, 2 Lathe 1 input queue overflow
;                            3 Mill 1 input queue overflow, 5 other fatal error, 6 ln negative

;These global variables are used to pass parameters between functions to execute the simulation
;Average of simulation results
AveInQ      .float 0.0  ;Average of number parts in queue
AveM1       .float 0.0  ;Average of machine 1 utilization
AveBlock    .float 0.0  ;Average block time of parts
AveR        .float 0.0  ;Average of robot utilization
AveStay     .float 0.0  ;Average time stay in system
AveDelay    .float 0.0  ;Average delay time of parts

NumInQ      .int 0      ;Number of parts waiting in M1 input queue
TotNum      .int 1      ;Total number of part entering
RoutedNum   .int 0      ;Record of routed part A from Lathe 1
SumPartA    .int 0      ;Total finished Part A
M1PState    .int 0      ;Previous state of machine 1 before broken

SumInQ      .float 0.0  ;Sum of number parts in queue
SumM1       .float 0.0  ;Sum of machine 1 utilization
SumDelay    .float 0.0  ;Total delay time of parts
SumBlock    .float 0.0  ;Total block time of parts
SumStay     .float 0.0  ;Total time stay in system
EndTime     .float 14400.0;End simulation time, 240 hours
MeArriA     .float 15.0 ;Mean time between arrival of Part A

Time        .float 0.0  ;Current simulating time, in minutes
LastTime    .float 0.0  ;Time of last event
NxtEvet     .int 0      ;Next event type, 0=empty, 1=arrival, 2=departure, 3=machine broken
;Event list
;current event
EveList0    .int 0      ;Event type = 0(current), 1(Arrival), 2(Departure), 3(sending), 4,
5(receive)
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 1      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
;arrival event
EveList1    .int 1      ;Event type
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 1      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
;departure event
EveList2    .int 2      ;Event type
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 1      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
```

```
;Part A sending to Mill 1 event
EveList3    .int 3      ;Event type
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 1      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time

;Workstation 1 breakdown event
EveList4    .int 4      ;Event type
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 1      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
;Part A receiving event
EveList5    .int 5      ;Event type
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 1      ;Which machine related to the event, 1(Lathe 1), 2(Mill 1)
            .float 0.0  ;Event happen time

InputQ0     .space 132  ;Input queue for the workstation 1, =6*21+6 for tail

;Defined the structure used in the program
;Describe workstation 1
M1          .int 1      ;Machine type, 1 = lathe, 2 = mill
            .int 0      ;Machine state = 0(idle), 1(busy), 2(blocked), 3(broken)
            .int 0      ;Current operating part type, 0 = Part A, 1 = Part B
            .float 6.0  ;Processing time for Part A
            .float 0.0  ;Processing time for Part B, not available
            .float 450  ;Mean time between failure
            .float 25   ;Mean time to repair

;Current part scheduled to be processed on machine
CurPart     .int 0      ;Part type = 0(Part A), 1(Part B)
            .int 1      ;Operation required on workstation 1, 0(No), 1(Yes)
            .int 1      ;Operation required on workstation 2, 0(No), 1(Yes)
            .float 0.0  ;Time when entering the FMS
            .float 0.0  ;Processing start time to one workstation
            .float 0.0  ;Finish time of one operation on one workstation

;Temporary storage of routed part
RoutedPart  .int 3      ;Part type = 0(Part A), 1(Part B), 3 (routed)
            .int 1      ;Operation required on Lathe 2, 0(No), 1(Yes)
            .int 1      ;Operation required on Mill 2, 0(No), 1(Yes)
            .float 0.0  ;Time when entering the FMS
            .float 0.0  ;Processing start time to one workstation
            .float 0.0  ;Finish time of one operation on one workstation


;Define the pointers to the arrays
PEveList    .word EveList0
PInputQ     .word InputQ0
PRNetwork   .word RNetwork2
PCurPart    .word CurPart
PRoutedPart .word RoutedPart
PM1_M2Part  .word M1_M2Part
PM1_M2Dev   .word M1_M2Dev
PGrephon    .word Grephon
PM1_M3Part  .word M1_M3Part
PSharedMem1 .word M2Finish
PSharedMem2 .word M1Down

*-------------------------------------------------------------------------
* DEFINE CONSTANTS
*-------------------------------------------------------------------------
QLimit      .int 20             ;Length limit of the queue
WIPLimit    .int 5              ;Limit of WIP queue
NumEvet     .int 5              ;Number of event
NumPType    .int 2              ;Number of part type

;Constant used by random number generation subroutine
A       .int    1078373         ; Constants needed for RAND
C       .int    2311527         ;
SEED    .int    0         ;
```

```
MASK    .word   0xFF7FFFFF       ;Mask for fast inverse float
MASK1   .word   0x007FFFFF       ;Mask for exponential distribution, %2^23
INVE23  .word   0xE9000000       ;1/2^23
BigNum  .float  1.0e29           ;Constant big value
        .float  1.0e30           ;

;Internal constants for ln(u)
;Scaling coefficients for ln(1+x)
LNRM    .float  0.6931471806     ; LN(2)
C0      .float  1.0000000000     ; C0 (1.0)
;       Polynomial coefficients for ln(1+X), 0 <= X < 1.
        .float  0.9999964239     ; TOP OF C1
        .float  -0.4998741238    ; TOP OF C2
        .float  0.3317990258     ; TOP OF C3
        .float  -0.2407338084    ; TOP OF C4
        .float  0.1676540711     ; TOP OF C5
        .float  -0.0953293897    ; TOP OF C6
        .float  0.0360884937     ; TOP OF C7
C8      .float  -0.0064535442    ; TOP OF C8
AC8     .word   C8

*-------------------------------------------------------------------------------
*Special signals for synchronization purpose, should be stored in the shared memory section 1,
*accessed by to both Microprocessor 1 and 2
*-------------------------------------------------------------------------------
    .sect "shared1"
M2Finish    .int 1      ;Indicate M2 finish the previous part departure event, *AR6
M2QFull     .int 0      ;Signal of WIP queue, 0 = not full, 1 = full, *+AR6(1)
PartArr     .int 0      ;Indicate Part A leave from Machine 1 to 2, *+AR6(2)
M2SafeTime  .float 0.0  ;Indicate how far simulation clock on M2 can advance, *+AR6(3)
M2D_Time    .float 0.0  ;Scheduled next part departure time on machine 2, *+AR6(4)

;Departure message from Machine 1 to Machine
M1_M2Dev    .int 0      ;Event type = 0(current), 1(Arrival), 2(Departure), 3(Machine Broken)
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 1      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
;Current Part transported from Machine 1 to Machine 2
M1_M2Part   .int 0      ;Part type = 0(Part A), 1(Part B)
            .int 1      ;Operation required on workstation 1, 0(No), 1(Yes)
            .int 1      ;Operation required on workstation 2, 0(No), 1(Yes)
            .float 0.0  ;Time when entering the FMS
            .float 0.0  ;Arrival time to one workstation
            .float 0.0  ;Finish time of one operation on one workstation

;Describe the robot
Grephon     .int 0      ;Robot state = 0(idle), 1(occupied)
            .int 1      ;current position of robot
            .float 0.0  ;Next available time of the robot
            .float 0.0  ;Next available time of the robot
            .float 0.0  ;Area of robot utilization

;Robot 1 path network, using real travel time if necessary
RNetwork2   .float 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
            .float 0.0, 0.0, 0.3, 0.6, 0.9, 1.2, 1.5, 0.3
            .float 0.0, 0.3, 0.0, 0.3, 0.6, 0.9, 1.2, 0.3
            .float 0.0, 0.6, 0.3, 0.0, 0.3, 0.6, 0.9, 0.6
            .float 0.0, 0.9, 0.6, 0.3, 0.0, 0.3, 0.6, 0.6
            .float 0.0, 1.2, 0.9, 0.6, 0.3, 0.0, 0.3, 0.9
            .float 0.0, 1.5, 1.2, 0.9, 0.6, 0.3, 0.0, 0.9
            .float 0.0, 0.3, 0.3, 0.6, 0.6, 0.9, 0.9, 0.0

SharedMem1  .word M2Finish
Barrier1    .int 0

*-------------------------------------------------------------------------------
*Special signals for synchronization purpose, should be stored in the shared memory section 2,
*accessed by to both Microprocessor 1 and 3
*-------------------------------------------------------------------------------
    .sect "shared2"
;Flags to synchronize the simulation of M1 and M2, indexed by AR5
```

```
M1Down      .int 0      ;Indicate Lathe 1 down, *AR5
PartArrival .int 0      ;Routed part A read by M2, 0=not, 1=read,2=no part, *+AR5(1)
PermitSend  .int 0      ;Part A allowed by M2 to be routed, 0=not, 1= allowed, *+AR5(2)
RequestSend .int 0      ;Request to send part A from Lathe 1 to Lathe 2, 0=no, 1=Yes, *+AR5(3)
M1SafeTime  .float 0.0  ;Indicate leading time on M1 during block, *+AR5(4)
M3SafeTime  .float 0.0  ;Indicate leading time on M3, *+AR5(5)
M3R_Time    .float 0.0  ;Scheduled next part routing time from Lathe 1 to Lathe 2, *+AR5(6)

;Part A transported from Lathe 1 to Lathe 2
M1_M3Part   .int 0      ;Part type = 0(Part A), 1(Part B)
            .int 1      ;Operation required on Lathe, 0(No), 1(Yes)
            .int 1      ;Operation required on Mill, 0(No), 1(Yes)
            .float 0.0  ;Time when entering the FMS
            .float 0.0  ;Arrival time to one workstation
            .float 0.0  ;Finish time of one operation on one workstation

SharedMem2  .word M1Down
Barrier2    .int 0

    .sect "code"
    .entry  _main
*---------------------------------------------------------------------------
* FUNCTION DEF : _main
* Control the advance of the program
*---------------------------------------------------------------------------
_main:
        LDP     @_main          ;Set Data Segment Pointer
        LDI     @STACK,SP       ;Set stack pointer

        LDI     @PSharedMem2,AR5 ;The index of flags in shared memory section 2, for
Microprocessor 3
        LDI     @PSharedMem1,AR6 ;The index of flags in shared memory section 1, for
Microprocessor 2

        CALL    _initial        ;Initialize the simulation, especially data structure
        CALL    _barrier

;Setup Timer 0 to count the execution time
        LDI     @BISR, R0
        STI     R0, @9FC9h      ;Save the branch instruction to Timer0 interrupt vector
        LDI     200h, R1
        STI     R1, @8020h      ;Using internal H1/2 clock for Timer 0
        LDI     @PERIOD, R1
        STI     R1, @8028h      ;Set period of Timer 0
        LDI     0,R1
        STI     R1,@8024h       ;Reset counter register
        LDI     2C0h, R1
        STI     R1, @8020h      ;Start Timer 0
        OR      @ETINT0, IE     ;Enable Timer 0 interrupt
        OR      @EINT, ST       ;Enable Interrupt

LM1     LDF     @Time,R0
        CMPF    @EndTime,R0     ;Does simulation end?
        BGT     LM2
        CALL    _timing         ;Determine the next event type
;Invoke the appropriate event handle function
        LDI     @PSUB,AR0
        ADDI    @NxtEvet,AR0
        LDI     *AR0,R0
        CALLU   R0              ;Handle part arrival, departure, or machine breakdown events
        B       LM1
LM2
        CALL    _report         ;Report simulation results

        LDI     0,R0
        STI     R0,@SysExit     ;Successfully run

;Calculate the execution time
        LDI     200h, R1
        STI     R1, @8020h      ;Halt the timer
        LDI     @PERIOD, R0     ;Calculate the whole cycles
```

```
        MPYI    @COUNT, R0
        LDI     @8024h, R1      ;Current counter number
        ADDI    R1, R0          ;R0 is the cycles
        STI     R0, @TOTCYC     ;Total cycle used

END     B       $               ;End of the program


*-------------------------------------------------------------------------
* FUNCTION DEF : _initial
* Fill the robot path network, initialize the data structures and the first arrival
* Input        : None
* Register     : R1, AR0, RC, RS, RE
* Output       : None
*-------------------------------------------------------------------------
_initial:
;Describe the current part on workstation 1, InputQ[0] is the storage of next part
        LDI     0,R1
        STI     R1,@InputQ0     ;Part A
        LDI     1,R1
        STI     R1,@InputQ0+1   ;Part A need turning operation on workstation 1
        STI     R1,@InputQ0+2   ;Part A need drilling operation on workstation 2
        LDF     @MeArriA,R0
        CALL    _expon
        STF     R0,@InputQ0+3   ;Part A first arrival time
        LDF     0.0,R1
        STF     R1,@InputQ0+4   ;Part A start on workstation 1, depending on robot
        STF     R1,@InputQ0+5   ;Time to finish operation on workstation 1 does not known yet

        LDF     @InputQ0+3,R1
        STF     R1,@EveList1+3  ;Schedule the arrive time

        LDF     @BigNum+1,R1
        STF     R1,@EveList2+3  ;Schedule the initial departure time to be infinite
        STF     R1,@EveList3+3  ;no sending now
        STF     R1,@EveList4+3  ;no machine down
        STF     R1,@EveList5+3  ;First route event time to be infinite

        RETS

*-------------------------------------------------------------------------
* FUNCTION DEF : _timing
* schedule the next event, synchronize and advance the simulation clock
* Input        : None
* Register     : R0, R1, R2, AR0, DP
* Output       : None
*-------------------------------------------------------------------------
_timing:
;Conservative parallel timing
        NOP
LT0     LDF     @BigNum,R1      ;A initial min time
        LDI     0,R0
        STI     R0,@NxtEvet     ;Next event type = 0
        LDI     1,R0            ;Iteration index
        LDI     @PEveList,AR0
        ADDI    3,AR0           ;Point to the first Event List E_Time
LT1     CMPF    *++AR0(4),R1    ;Event List[i].E_Time <= min_time?
        BLT     LT2
        LDF     *AR0,R1         ;Minimum time is R1
        STI     R0,@NxtEvet     ;Next Event No.
LT2     ADDI    1,R0
        CMPI    5,R0            ;Need to compare all 5 events
        BLE     LT1
;Check to see if the event can be safely processed
LT3     LDF     @EveList3+3,R0
        CMPF    @BigNum,R0
        BLT     LT4             ;Check if next sending time is set to be infinite
        LDF     R1,R0           ;If yes, set M2 safe time to be current time
LT4     STF     R0,*+AR6(3)     ;Safe time for M4 is next departure time
                                ;Microprocessor 4 can process events before this M2safetime
        STF     R1,*+AR5(4)     ;Set M1 safe time during broken period as Lathe 2 current time
```

```
        CMPF    *+AR5(5),R1     ;Is time <= M2 Safe Time?
        BGT     LT0             ;No, recheck the safe time again

        LDI     @NxtEvet,R0     ;If event list is empty
        BNZ     LT8
        LDI     1,R0            ;Exit(1), event list is empty
        CALL    _exit           ;If even list is empty, stop the simulation and exit

;All situtations
LT8     STF     R1,@Time        ;Update Simulation clock
        LDF     @LastTime,R0    ;Last event time
        SUBF3   R0,R1,R2        ;Compute time since last event
        STF     R1, @LastTime   ;Save last event time

        FLOAT   @NumInQ,R0      ;Statistical update
        MPYF    R2,R0
        ADDF    @SumInQ,R0
        STF     R0,@SumInQ      ;Update area of number in M1 queue

        RETS



*---------------------------------------------------------------------------
* FUNCTION DEF : _arrive
* schedule the next arrival event and handle current arrival event
* Input        : None
* Register     : R0, R1, R2, AR0, AR1
* Output       : None
*---------------------------------------------------------------------------
_arrive:
        LDI     @M1+1,R0        ;Check to see if workstation 1 is idle?
        BNZ     LA1             ;If not idle, go to LA1
        LDI     1,R0            ;If idle
        STI     R0,@M1+1        ;Set Machine 1 busy
        LDI     2,R1            ;Request robot, Load part from queue to machine 1
        CALL    _rerobot        ;R0 = initial position, R1 = destination position
                                ;Travel time return in R0
        LDF     @Time,R1
        STF     R1,@CurPart+3   ;Current part entering time
        ADDF3   R0,R1,R2
        STF     R2,@CurPart+4   ;Current process starting time
        LDF     @M1+3,R0
        CALL    _expon          ;Processing time for the part
        ADDF    @CurPart+4,R0
        STF     R0,@EveList2+3  ;Schedule a Part 1 departure after processing
                                ;considering the travel time of robot
        STF     R0,@EveList3+3  ;Sending event
        STF     R0,@CurPart+5   ;Current part finishing time
        ADDF    @SumM1,R0
        SUBF    @CurPart+4,R0
        STF     R0,@SumM1       ;Update Machine busy period
        B       LA4
*** B   LA4 ;BRANCH OCCURS
;If machine is not available (either busy, blocked or broken), put the current part in queue
LA1     LDI     @NumInQ,R1
        ADDI    1,R1
        STI     R1,@NumInQ      ;Increase the length of queue
        CMPI    20,R1
        BLE     LA2
        LDI     2,R0            ;Queue is overflow, exit 2
        CALL    _exit
LA2     LDI     @NumInQ,AR0     ;InputQ[NumInQ] = InputQ[0]
        MPYI    6,AR0
        ADDI    @PInputQ,AR0
        LDI     @PInputQ,AR1
        LDI     5,RC
        RPTB    LA3
        LDI     *AR1++,R0       ;
LA3     STI     R0,*AR0++
;Schedule next Part A arrival
```

```
LA4     LDF     @MeArriA,R0
        CALL    _expon
        ADDF    @Time,R0
        STF     R0,@InputQ0+3
        STF     R0,@EveList1+3  ;Next arrival time
        LDI     @TotNum,R1      ;Update total entering part number
        ADDI    1,R1
        STI     R1,@TotNum
        RETS

*--------------------------------------------------------------------------
* FUNCTION DEF : _depart
* handle current departure event
* Input        : None
* Register     : R0, R1, R2, R3, R4, R5, AR0, AR1, RC, RS, RE, DP
* Output       : None
*--------------------------------------------------------------------------
_depart:
        NOP
;Update part A stay in system
        LDF     @SumStay,R2
        ADDF    @CurPart+5,R2
        SUBF    @CurPart+3,R2
        STF     R2,@SumStay
;Schedule next part departure
        LDI     @NumInQ ,R3     ;Is the input queue empty?
        BNZ     LD4             ;No, go to LD4
        LDI     0,R1
        STI     R1,@M1+1        ;Yes, set machine 1 idle, R1=0
        LDF     @BigNum+1,R3
        STF     R3,@EveList2+3  ;No next departure
        STF     R3,@EveList3+3  ;No sending
        B       LD8
*** B   LD5  ;BRANCH OCCURS
LD4     LDI     1, R0
        STI     R0,@M1+1        ;Machine busy R0 = 1
        LDI     @NumInQ,R3      ;Reduce queue length
        SUBI    R0,R3
        STI     R3,@NumInQ
        LDF     @Time,R5        ;Update delay accumulator
        SUBF    @InputQ0+9,R5
        LDF     @SumDelay,R3
        ADDF    R5,R3
        STF     R3,@SumDelay

        LDI     @PInputQ,AR0
        ADDI    6,AR0
        LDI     @PCurPart,AR1
        LDI     5,RC
        RPTB    LD6
        LDI     *AR0++,R0       ;CurPart = InputQ[1]
LD6     STI     R0,*AR1++

;Maintain the queue
        LDI     @PInputQ,AR0
        ADDI    6,AR0           ;First in queue
        LDI     6,R0            ;
        ADDI3   R0,AR0,AR1      ;Next in queue
        LDI     @NumInQ,R0
        ADDI    1,R0            ;Fill the last position with zeros
        MPYI    6,R0            ;The total number need to be relocated
        SUBI    1,R0            ;
        LDI     R0,RC           ;Repeat (NumInQ+1)*6 times
        RPTB    LD7
        LDI     *AR1++,R1
LD7     STI     R1,*AR0++

;Schedule next departure
        LDI     2,R0
        LDI     3,R1
        CALL    _rerobot        ;Request robot
```

```
        ADDF    @Time,R0
        STF     R0,@CurPart+4
        LDF     @M1+3,R0
        CALL    _expon          ;Processing time
        ADDF    @CurPart+4,R0
        STF     R0,@EveList2+3  ;Next departure time
        STF     R0,@EveList3+3  ;Next sending time
        STF     R0,@CurPart+5   ;Part finished time
        ADDF    @SumM1,R0       ;Update Machine busy period
        SUBF    @CurPart+4,R0
        STF     R0,@SumM1

LD8     LDI     *+AR5(3),R0     ;Check if Lathe 1 request to route parts
        CMPI    1,R0
        BNZ     LDE             ;If no request, return
        LDI     @M1+1,R0
        BZ      LD9             ;If Lathe 2 is idle, schedule a sending-receiving
        LDI     @CurPart,R0
        CMPI    3,R0            ;If Lathe 2 is not idle, but next part is routed from Lathe 1
        BNZ     LD10            ;Set not permit send

LD9     LDF     @Time,R0        ;Yes, schedule a part routing
        STF     R0,@EveList5+3  ;Schedule next receiving
        STF     R0,*+AR5(6)     ;Schedule next sending event on M1
        LDI     1,R1
        STI     R1,*+AR5(2)     ;Permit M1 to route another part A
        B       LDE
LD10    LDI     0,R1
        STI     R1,*+AR5(2)     ;Not permit M1 to route

LDE     RETS


*---------------------------------------------------------------------------
* FUNCTION DEF : _send
* handle route part A from Lathe 2 to Mill 2
* Input        : None
* Register     : R0, R1, R2, R3,AR0, AR1
* Output       : None
*---------------------------------------------------------------------------
_send:
        NOP
LS0     LDI     *AR6,R0         ;Check M2Finish to see if previous departure has been processed
or not?
        CMPI    1,R0
        BNZ     LS0             ;If not, wait until microprocessor 4 processed previous departure

        LDI     *+AR6(1),R0
        CMPI    1,R0            ;Check to see if Mill 1 WIP queue is full
        BNZ     LS1
        LDF     *+AR6(4),R1
        CMPF    @EveList3+3,R1  ;If Departure time < M2D_Time?
        BLE     LSE             ;No, wait until M2 process the next departure
        LDF     R1,R5           ;Yes, save the possible departure time
        SUBF    @Time,R5        ;Blocked time period
        LDF     @SumBlock,R0
        ADDF    R5,R0
        STF     R0,@SumBlock    ;Update the total block accumulator
        STF     R1,@CurPart+5   ;Update the finished time for current part
;Reschedule the current departure to the new possible departure time
        STF     R1,@EveList2+3
        STF     R1,@EveList3+3
        LDI     2,R1            ;Machine 2 WIP queue full, M1 blocked
        STI     R1,@M1+1
        B       LSE
;Schedule part A arrival on Mill 1
LS1     LDI     @EveList3,R0    ;M1_M2Dev = EveList3
        LDI     @EveList3+1,R1
        LDI     @EveList3+2,R2
        LDF     @EveList3+3,R3
        LDI     @PM1_M2Dev,AR0  ;Point to the shared memory
```

```
        STI     R0,*AR0
        STI     R1,*+AR0(1)
        STI     R2,*+AR0(2)
        STF     R3,*+AR0(3)
        LDI     0,R1
        STI     R1,*AR6         ;Event not processed by microprocessor 4
;Machine is up. Schedule a Part A leave from M1 to M2.
;Wait until microprocessor 2 read the previous messages

LS2     LDI     *+AR6(2),R0
        CMPI    1,R0            ;Wait until microprocessor 4 read previous message, check PartArr
flag
        BZ      LS2
        LDI     @PCurPart,AR0  ;M1_M2Par = CurPart
        LDI     @PM1_M2Part,AR1 ;Point to the shared memory
        LDI     5,RC
        RPTB    LS3
        LDI     *AR0++,R0       ;SendPart = CurPart
LS3     STI     R0,*AR1++
        LDI     *-AR0(1),R0
        STI     R0,*-AR1(3)     ;Reset the starting time on Mill 2
        LDI     1,R0
        STI     R0,*+AR6(2)     ;Inform machine 2 part A arrival

LS4     LDF     @BigNum+1,R0
        STF     R0,@EveList3+3  ;Reset sending event
        LDI     @SumPartA,R3    ;Part A throughput
        ADDI    1,R3
        STI     R3,@SumPartA
LSE     RETS


*-----------------------------------------------------------------------
* FUNCTION DEF : _down
* no break down on Lathe 2
* Input        : None
* Register     : None
* Output       : None
*-----------------------------------------------------------------------
_down:
        NOP
M1LOE   RETS

*-----------------------------------------------------------------------
* FUNCTION DEF : _receive
* handle when machine break down receiving the routed part A from Lathe 2 to Lathe 2
* Input        : None
* Register     : R0, R1, R2, R3,R4,R5,AR0,AR1
* Output       : None
*-----------------------------------------------------------------------
_receive:
        LDP     @_main
M1RO0   LDI     *+AR5(1),R0
        CMPI    0,R0
        BZ      M1RO0           ;Wait until Lathe 1 give a signal
        CMPI    1,R0            ;Check if a new part routed
        BZ      M1RO1           ;1 means a part routed, 2 means nor part routed
        LDI     0,R0
        STI     R0,*+AR5(1)     ;Clear the Signal
        B       M1RO10
M1RO1   LDI     @PM1_M3Part,AR0 ;Point to the shared memory of routed part
        LDI     *AR0,R0         ;Get the routed part
        LDI     *+AR0(1),R1
        LDI     *+AR0(2),R2
        LDF     *+AR0(3),R3
        LDF     *+AR0(4),R4
        LDF     *+AR0(5),R5
        LDI     @PRoutedPart,AR0 ;Point to the temporary storage of routed part
        LDI     3,R0             ;Marked the part type as routed
        STI     R0,*AR0          ;Receive the routed part
        STI     R1,*+AR0(1)
```

```
        STI     R2,*+AR0(2)
        STF     R3,*+AR0(3)
        STF     R4,*+AR0(4)
        STF     R5,*+AR0(5)
        LDI     0,R0
        STI     R0,*+AR5(1)      ;Set routed part has been read by M2
;Increase number of routed part by 1
        LDI     @RoutedNum,R0    ;Increase routed part by 1
        ADDI    1,R0
        STI     R0,@RoutedNum;
;Insert the routed part into the queue of Lathe 2
;Find the position in queue using AT rule
        LDI     0,R2             ;Iteration index
        LDI     @PInputQ,AR0
        ADDI    9,AR0            ;First arrival time in queue
M1RO3   ADDI    1,R2             ;Increase index
        LDF     *AR0++(6),R0     ;The arriving time of parts in queue
        CMPF    @PRoutedPart+3,R0;Compare arriving time of routed part to that of parts in queue
        BLE     M1RO3            ;R2 store the position routed part should be put in queue
;Insert into queue, position stored in R2
        CMPI    @QLimit,R2       ;Check if queue is overflow?
        BLE     M1RO4
        LDI     4,R1             ;Queue is overflow, exit 4
        CALL    _exit
M1RO4   LDI     @NumInQ,R0
        ADDI    1,R0
        CMPI    R0,R2
        BZ      M1RO7            ;If insert position at the end of queue, need not move
;Move parts in queue
        MPYI    6,R0             ;
        SUBI    1,R0
        LDI     @PInputQ,AR0
        ADDI    R0,AR0           ;The end position of part to be moved
        LDI     AR0,AR1
        ADDI    6,AR1            ;The first position begin to move
;Calculate total number to move
        LDI     @NumInQ,R0    ;
        ADDI    1,R0
        SUBI    R2,R0            ;Total parts need to be moved = M1NumInQ+1-InsertPosition
        MPYI    6,R0
        SUBI    1,R0
        LDI     R0,RC            ;Repeat (NumInQ+1-InsertPosition)*6 times
        RPTB    M1RO5
        LDI     *AR0--,R1
M1RO5   STI     R1,*AR1--        ;Move parts up
;Insert from temporary memory into queue
        ADDI    1,AR0            ;Insert position in AR0 after moving
        B       M1RO8

M1RO7   LDI     @PInputQ,AR0
        LDI     @NumInQ,R0
        MPYI    6,R0
        ADDI    R0,AR0
        ADDI    6,AR0             ;Insert position in AR0, for insert at end of queue
M1RO8   LDI     @PRoutedPart,AR1 ;Insert routed part into queue
        LDI     5,RC
        RPTB    M1RO9
        LDI     *AR1++,R1
M1RO9   STI     R1,*AR0++

        LDI     @NumInQ,R0    ;Increase queue length
        ADDI    1,R0
        STI     R0,@NumInQ

M1RO10  LDF     @BigNum+1,R1    ;Set next receiving event time to be infinite
        STF     R1,@EveList5+3
M1ROE   RETS


*---------------------------------------------------------------------
* FUNCTION DEF : _rerobot
```

```
* handle robot request, return robot travel time.
* Input        : R0,initial position, R1, destination position
* Register     : R0, R1, AR0, AR1, IR0, IR1, (only AR0,AR1, IR0,IR1 restored)
* Output       : R0 travel time
*-------------------------------------------------------------------------
_rerobot:
;Calculate travel time, robot need to travel from current position to initial position,
;then from initial position to the destination
        PUSH    AR0
        PUSH    AR1
        PUSH    IR0
        PUSH    IR1

        LDI     @PGrephon, AR1          ;Pointer to the robot
        LDI     *+AR1(1),IR0
        MPYI    8,IR0
        ADDI    R0,IR0                  ;From current position to starting position
        LDI     @PRNetwork,AR0
        LDI     R0,IR1
        MPYI    8,IR1
        ADDI    R1,IR1                  ;From starting position to destination
;Robot traveltime = RNetwork[Grephon.R_Pos][initpos] + RNetwork[initpos][destpos]
        ADDF3   *+AR0(IR1),*+AR0(IR0),R0
        STI     R1,*+AR1(1)             ;Next robot position
        LDF     @Time,R1
        CMPF    *+AR1(2),R1
        BGE     LR1
        ADDF    *+AR1(2),R0             ;Robot is occupied, considering some waiting time
        SUBF    R1,R0
;Next time robot stay at initial position and available time is current time plus travel time
LR1     ADDF    R0,R1                   ;Next available time
        STF     R1,*+AR1(2)
        LDF     2.0,R1
        MPYF    R0,R1
        ADDF    *+AR1(4),R1             ;Update robot utilization
        STF     R1,*+AR1(4)

        POP     IR1
        POP     IR0
        POP     AR1
        POP     AR0

        RETS

*-------------------------------------------------------------------------
* FUNCTION DEF : _report
* report simulation results
* Input        : None
* Register     : R0, R2, R5, AR0
* Output       : None
*-------------------------------------------------------------------------
_report:
        LDF     @EndTime,R0
        CALL    INV_F30
        RND     R0,R5           ;R5=1/Time

        LDF     @SumM1,R0
        MPYF    R5,R0
        STF     R0,@AveM1       ;Utilization of M1

        LDI     @PGrephon,AR0
        LDF     *+AR0(4),R0
        MPYF    R5,R0
        STF     R0,@AveR        ;Utilization of robot

        LDF     @SumBlock,R0
        MPYF    R5,R0
        STF     R0,@AveBlock    ;Average blocked

        LDF     @SumInQ,R0
        MPYF    R5,R0
```

```
        STF     R0,@AveInQ      ;Average waiting in queue

        FLOAT   @SumPartA,R0
        CALL    INV_F30
        RND     R0,R2           ;R2 = 1/SumPart
        LDF     @SumDelay,R0
        MPYF    R2,R0
        STF     R0,@AveDelay    ;Average delay

        LDF     @SumStay,R0
        MPYF    R2,R0
        STF     R0,@AveStay     ;Average stay in FMS
        RETS

*---------------------------------------------------------------------
* FUNCTION DEF : _barrier
* Unusually exit
* Input        : none
* Register     : R0
* Output       : none
*---------------------------------------------------------------------
_barrier:
        LDP     SharedMem1
        LDI     @Barrier1,R0
        ADDI    1,R0
        STI     R0,@Barrier1
B1      LDI     @Barrier1,R0
        CMPI    2,R0
        BLT     B1
        LDP     SharedMem2
        LDI     @Barrier2,R0
        ADDI    1,R0
        STI     R0,@Barrier2
B2      LDI     @Barrier2,R0
        CMPI    2,R0
        BLT     B2
        LDP     @_main
        RETS

*---------------------------------------------------------------------
* FUNCTION DEF : _exit
* Unusually exit
* Input        : none
* Register     : none
* Output       : none
*---------------------------------------------------------------------
_exit:
        LDP     @_main
        STI     R0,@SysExit
EXITSIM B       $

*---------------------------------------------------------------------
* FUNCTION DEF : _expon
* Generate a random number with exponential distribution
* Input        : R0 is the mean
* Register     : R1
* Output       : R0 is the random number with exponential distribution
*---------------------------------------------------------------------
_expon:
*Fast 32 bit uniform random number generator
        PUSH    R1
        PUSHF   R1

        NEGF    R0, R1          ; R1 is the negative of the mean
        PUSH    R1
        PUSHF   R1              ; Save -mean in the stack, all 40 bits
        LDI     @SEED,R0        ; Call here for last SEED
        MPYI    @A,R0           ; Calculate r = (A*r+C)%m
        ADDI    @C,R0           ;
        AND     @MASK1, R0      ; m=2^23
        STI     R0,@SEED        ; Result is returned
```

```
        ADDI    1,R0            ;
        FLOAT   R0,R0           ;
        MPYF    @INVE23,R0      ; r/m
        CALL    _ln             ; ln(u)
        POPF    R1              ; -Mean
        POP     R1
        MPYF    R1, R0          ; random number =  -mean*ln(u)

        POPF    R1
        POP     R1
        RETS


*-------------------------------------------------------------------------
*Define references
*-------------------------------------------------------------------------
*Define the entry point of subroutines
SUB     .word _main
        .word _arrive
        .word _depart
        .word _send
        .word _down
        .word _receive
PSUB    .word SUB

        .sect "isrcode"
*Interrupt Service Routine
_ISRCount:
        PUSH DP
        PUSH R0
        PUSHF R0
        LDP _main
        LDI @COUNT, R0
        ADDI 1, R0
        STI R0, @COUNT
        POPF R0
        POP R0
        POP DP
        RETI

STACK   .word  $+1          ; Bottom of the Stack
        .end
```

Note:

1. Subroutines _ln and _INVF30 are the same as in 1MhardL1.asm

C.3.2.4 Code running on DSP 4

```
*-------------------------------------------------------------------
* 4MHardM2.ASM -- conservative parallel discrete event simulation
* of a scenario: 4 machine, 2 part type, 2 robot, and considering machine breakdown
* This is the program running on Microprocessor 4, using to simulating Mill 2
* Programmed by: Dong Xu
* Date         : Mar.22, 2001
* Modified     : Apr.3, 2001
*-------------------------------------------------------------------
        .start "data", 0x809900    ;Data section, 809900-809AFF
        .start "code", 0x809B00     ;Code section, 809B00-809EFF
        .start "shared1", 0xA00000   ;Shared memory section, A00000-A003FF
        .start "text", 0xB00000    ;Main text section, B00000-B007FFF

        .sect "data"
;These global variables are used to pass parameters between functions to execute the simulation
;Average of simulation results
SysExit     .int 5      ;Exit state of the simulation
;                            0 success, 1 event list empty, 2 Lathe 1 input queue overflow
;                            3 Mill 1 input queue overflow, 5 other fatal error, 6 ln negative
AveInQ      .float 0.0  ;Average of number parts in queue for part A
            .float 0.0  ;Part B
AveM2       .float 0.0  ;Average of machine 2 utilization
AveR        .float 0.0  ;Average of robot utilization
AveStay     .float 0.0  ;Average time stay in system
            .float 0.0
AveDelay    .float 0.0  ;Average delay time of parts
            .float 0.0

NumInQ      .int 0      ;Number of parts waiting in M2 input queue
            .int 0
TotNum      .int 0      ;Total number of parts entering
            .int 1
SumPart     .int 0      ;Total finished Parts
            .int 0

SumInQ      .float 0.0  ;Sum of number parts in queues
            .float 0.0
SumM2       .float 0.0  ;Sum of machine 2 utilization
SumDelay    .float 0.0  ;Total delay time of parts
            .float 0.0
SumStay     .float 0.0  ;Total time stay in system
            .float 0.0
EndTime     .float 14400.0;End simulation time, 240 hours
MeArriB     .float 20.0 ;Mean time between arrival of Part B

Time        .float 0.0  ;Current simulating time, in minutes
LastTime    .float 0.0  ;Time of last event
NxtEvet     .int 0      ;Next event type, 0=empty, 1=part B arrival, 2=part A arrival
                        ;3=parts departure
;Event list
;current event
EveList0    .int 0      ;Event type = 0(current), 1(Arrival1), 2(Arrival2), 3(Departure)
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 2      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
;Part B arrival event
EveList1    .int 1      ;Event type = 0(current), 1(Arrival1), 2(Arrival2), 3(Departure)
            .int 1      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 2      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
;Part A arrival event
EveList2    .int 2      ;Event type = 0(current), 1(Arrival1), 2(Arrival2), 3(Departure))
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 2      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
;departure event
EveList3    .int 3      ;Event type = 0(current), 1(Arrival1), 2(Arrival2), 3(Departure)
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 2      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
```

```
WIPQ0       .space 42   ;Working-in-process queue for part A, = 6*6+6 = 42
InputQ0     .space 132  ;Input queue for part B, =6*21+6 for tail

;Defined the structure used in the program
;Describe workstation 2
M2          .int 2      ;Machine type, 1 = lathe, 2 = mill
            .int 0      ;Machine state = 0(idle), 1(busy), 2(blocked), 3(broken)
            .int 0      ;Current operating part type, 0 = Part A, 1 = Part B
            .float 8.0  ;Processing time for Part A
            .float 4.0  ;Processing time for Part B, not available
            .float 0.0  ;Mean time between failure
            .float 0.0  ;Mean time to repair

;Current part scheduled to be processed on machine
CurPart     .int 1      ;Part type = 0(Part A), 1(Part B)
            .int 0      ;Operation required on workstation 1, 0(No), 1(Yes)
            .int 1      ;Operation required on workstation 2, 0(No), 1(Yes)
            .float 0.0  ;Time when entering the FMS
            .float 0.0  ;Processing start time to one workstation
            .float 0.0  ;Finish time of one operation on one workstation

;Define the pointers to the arrays
PAveInQ     .word AveInQ
PAveDelay   .word AveDelay
PAveStay    .word AveStay
PNumInQ     .word NumInQ
PTotNum     .word TotNum
PSumInQ     .word SumInQ
PSumPart    .word SumPart
PSumDelay   .word SumDelay
PSumStay    .word SumStay
PM2         .word M2
PCurPart    .word CurPart
PEveList    .word EveList0
PInputQ     .word InputQ0
PWIPQ       .word WIPQ0
PRNetwork   .word RNetwork1
PM1_M2Part  .word M1_M2Part
PM1_M2Dev   .word M1_M2Dev
PGrephon    .word Grephon
PSharedMem1 .word M2Finish

*-------------------------------------------------------------------------
* DEFINE CONSTANTS
*-------------------------------------------------------------------------
QLimit      .int 20             ;Length limit of the queue
WIPLimit    .int 5              ;Limit of WIP queue
NumEvet     .int 3              ;Number of event
NumPType    .int 2              ;Number of part type

;Constant used by random number generation subroutine
A       .int    1078373         ; Constants needed for RAND
C       .int    2311527         ;
SEED    .int    0       ;
MASK    .word   0xFF7FFFFF      ;Mask for fast inverse float
MASK1   .word   0x007FFFFF      ;Mask for exponential distribution, %2^23
INVE23  .word   0xE9000000      ;1/2^23
BigNum  .float  1.0e29          ;Constant big value
        .float  1.0e30          ;

;Internal constants for ln(u)
;Scaling coefficients for ln(1+x)
LNRM    .float  0.6931471806    ; LN(2)
C0      .float  1.0000000000    ; C0 (1.0)
;       Polynomial coefficients for ln(1+X), 0 <= X < 1.
        .float  0.9999964239    ; TOP OF C1
        .float  -0.4998741238   ; TOP OF C2
        .float  0.3317990258    ; TOP OF C3
        .float  -0.2407338084   ; TOP OF C4
        .float  0.1676540711    ; TOP OF C5
```

```
        .float  -0.0953293897    ; TOP OF C6
        .float  0.0360884937     ; TOP OF C7
C8      .float  -0.0064535442    ; TOP OF C8
AC8     .word   C8

*-------------------------------------------------------------------------------
*Special signals for synchronization purpose, should be stored in the shared memory,
*accessed by to both Microprocessor
*-------------------------------------------------------------------------------
    .sect "shared1"
M2Finish    .int 1      ;Indicate M2 finish the previous part departure event, *AR6
M2QFull     .int 0      ;Signal of WIP queue, 0 = not full, 1 = full, *+AR6(1)
PartArr     .int 0      ;Indicate Part A leave from Machine 1 to 2, *+AR6(2)
M2SafeTime  .float 0.0  ;Indicate how far simulation clock on M2 can advance, *+AR6(3)
M2D_Time    .float 0.0  ;Scheduled next part departure time on machine 2, *+AR6(4)

;Departure message from Machine 1 to Machine
M1_M2Dev    .int 0      ;Event type = 0(current), 1(Arrival), 2(Departure), 3(Machine Broken)
            .int 0      ;Which part related to the event, 0(Part A), 1(Part B)
            .int 1      ;Which machine related to the event, 1(Workstation 1), 2(Workstation 2)
            .float 0.0  ;Event happen time
;Current Part transported from Machine 1 to Machine 2
M1_M2Part   .int 0      ;Part type = 0(Part A), 1(Part B)
            .int 1      ;Operation required on workstation 1, 0(No), 1(Yes)
            .int 1      ;Operation required on workstation 2, 0(No), 1(Yes)
            .float 0.0  ;Time when entering the FMS
            .float 0.0  ;Arrival time to one workstation
            .float 0.0  ;Finish time of one operation on one workstation

;Describe the robot
Grephon     .int 0      ;Robot state = 0(idle), 1(occupied)
            .int 1      ;current position of robot
            .float 0.0  ;Next available time of the robot
            .float 0.0  ;Next available time of the robot
            .float 0.0  ;Area of robot utilization

;Robot 1 path network, using real travel time if necessary
RNetwork1   .float 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
            .float 0.0, 0.0, 0.3, 0.6, 0.9, 1.2, 1.5, 0.3
            .float 0.0, 0.3, 0.0, 0.3, 0.6, 0.9, 1.2, 0.3
            .float 0.0, 0.6, 0.3, 0.0, 0.3, 0.6, 0.9, 0.6
            .float 0.0, 0.9, 0.6, 0.3, 0.0, 0.3, 0.6, 0.6
            .float 0.0, 1.2, 0.9, 0.6, 0.3, 0.0, 0.3, 0.9
            .float 0.0, 1.5, 1.2, 0.9, 0.6, 0.3, 0.0, 0.9
            .float 0.0, 0.3, 0.3, 0.6, 0.6, 0.9, 0.9, 0.0

SharedMem1  .word M2Finish
Barrier1    .int 0

    .sect "code"
    .entry  _main
*-------------------------------------------------------------------------------
* FUNCTION DEF : _main
* Control the advance of the program
*-------------------------------------------------------------------------------
_main:
        LDP     @_main          ;Set Data Segment Pointer
        LDI     @STACK,SP       ;Set stack pointer
        LDI     @PSharedMem1,AR6

        CALL    _initial        ;Initialize the simulation, especially data structure
        CALL    _barrier        ;barrier for starting

LM1     LDF     @Time,R0
        CMPF    @EndTime,R0     ;Does simulation end?
        BGT     LM2

        CALL    _timing         ;Determine the next event type

;Invoke the appropriate event handle function
        LDI     @PSUB,AR0
```

```
        ADDI    @NxtEvet,AR0
        LDI     *AR0,R0
        CALLU   R0              ;Handle parts arrival, departure
        B       LM1
LM2
        CALL    _report         ;Report simulation results

        LDI     0,R0
        STI     R0,@SysExit     ;Successfully run

END     B       $               ;End of the program


*-------------------------------------------------------------------------
* FUNCTION DEF : _initial
* Fill the robot path network, initialize the data structures and the first arrival
* Input        : None
* Register     : R1, AR0, RC, RS, RE
* Output       : None
*-------------------------------------------------------------------------
_initial:
;Describe the current part on workstation 2, WIPQ0[0] is the storage of next part A
        LDI     0,R1
        STI     R1,@WIPQ0       ;Part A
        LDI     1,R1
        STI     R1,@WIPQ0+1     ;Part A need turning operation on workstation 1
        STI     R1,@WIPQ0+2     ;Part A need drilling operation on workstation 2
        LDF     0.0,R1
        STF     R1,@WIPQ0+3     ;Part A first arrival time
        STF     R1,@WIPQ0+4     ;Part A start on workstation 2
        STF     R1,@WIPQ0+5     ;Time to finish operation on workstation 2
;Describe the part B on workstation 2, InputQ0[0] is the storage of next part
        LDI     1,R0
        STI     R0,@InputQ0     ;Part B
        LDI     0,R1
        STI     R1,@InputQ0+1   ;Part B do not need turning operation on workstation 1
        STI     R0,@InputQ0+2   ;Part B need drilling operation on workstation 2
        LDF     @MeArriB,R0
        CALL    _expon
        STF     R0,@InputQ0+3   ;Part B first arrival time
        LDF     0.0,R1
        STF     R1,@InputQ0+4   ;Part B start on workstation 2, depending on robot
        STF     R1,@InputQ0+5   ;Time to finish operation on workstation 2 does not known yet
;Schedule first arrivals
        LDF     @InputQ0+3,R1
        STF     R1,@EveList1+3  ;Schedule the part B arrive time
        LDF     @BigNum+1,R1
        STF     R1,@EveList2+3  ;The arrival time of part A is passed by Microprocessor 1
        STF     R1,@EveList3+3  ;Schedule the initial departure time to be infinite
        RETS

*-------------------------------------------------------------------------
* FUNCTION DEF : _timing
* schedule the next event, synchronize and advance the simulation clock
* Input        : None
* Register     : R0, R1, R2, R3, R4, R5, AR0, AR1, DP, RC, RS, RE
* Output       : None
*-------------------------------------------------------------------------
_timing:
        NOP
LT1     LDI     *+AR6(2),R0     ;Check for PartArr flag
        CMPI    1,R0            ;Synchronize with microprocessor 1, check for new part A arrival
        BNZ     LT15
;Schedule a part A arrival
        LDI     @PM1_M2Dev,AR0  ;Point to M1_M2Dev
        LDI     *AR0,R0         ;Event List 2 = M1_M2Dev
        LDI     *+AR0(1),R1
        LDI     *+AR0(2),R2
        LDF     *+AR0(3),R3
        STI     R0,@EveList2
        STI     R1,@EveList2+1
```

```
        STI     R2,@EveList2+2
        STF     R3,@EveList2+3

;Determine the minimum time among event list and set the next event type
;If two event happen at the same time, handle machine broken event first, then
;departure event, and the last is arrival event
LT15    LDF     @BigNum,R1      ;A initial min time
        LDI     0,R0
        STI     R0,@NxtEvet     ;Next event type = 0
        LDI     1,R0            ;Iteration index
        LDI     @PEveList,AR0
        ADDI    3,AR0           ;Point to the first Event List E_Time
LT2     CMPF    *++AR0(4),R1    ;Event List[i].E_Time <= min_time?
        BLT     LT3
        LDF     *AR0,R1         ;Minimum time is
        STI     R0,@NxtEvet     ;Next Event No.
LT3     ADDI    1,R0
        CMPI    3,R0            ;Need to compare all 3 events
        BLE     LT2

        LDI     @NxtEvet,R0     ;If event list is empty
        BNZ     LT4
        LDI     1,R0            ;Exit(1), event list is empty
        CALL    _exit           ;If even list is empty, stop the simulation and exit
LT4     CMPF    @EndTime, R1
        BGT     LT5             ;If Time > simulation end time, then jump out
        CMPF    *+AR6(3),R1     ;Check if can safely process the next event
        BGT     LT1             ;Workstation 2 cannot process event has time stamp
                                ;larger than the SafeTime.
                                ;SafeTime must be provided by microprocessor 1
LT5     STF     R1,@Time        ;Simulation clock
        LDF     @LastTime,R0    ;Last event time
        SUBF3   R0,R1,R2        ;Compute time since last event
        STF     R1, @LastTime   ;Save last event time

        LDI     @PNumInQ,AR0    ;Pointer of NumInQ
        LDI     @PSumInQ,AR1    ;Pointer SumInQ
        LDI     1,RC            ;Repeat 2 times
        RPTB    LT6
        FLOAT   *AR0++,R0
        MPYF    R2,R0
        ADDF    *AR1,R0
LT6     STF     R0,*AR1++       ;Update area of number in queues
LTE     RETS

*---------------------------------------------------------------------------
* FUNCTION DEF : _arriveb
* schedule the next arrival event and handle current arrival event
* Input        : None
* Register     : R0, R1, R2, R4, AR0, AR1, RC, RS, RE
* Output       : None
*---------------------------------------------------------------------------
_arriveb:
;Check to see if Mill 1 is idle
        LDI     @M2+1,R0
        BNZ     LA4             ;M2 is not idle

        LDI     1,R0            ;M2 is idle, set it to busy
        STI     R0,@M2+1
;Load part from queue to machine 2, WIP to M2 is 3->5, Input Queue to M2 is 4->5,
        LDI     5,R1            ;Destination
        LDI     4,R0            ;Starting
        CALL    _rerobot        ;Travel time return in R0
        LDF     @Time,R1
        STF     R1,@CurPart+3   ;Entering time
        ADDF    R0,R1
        STF     R1,@CurPart+4   ;Start processing time
        LDI     @PM2,AR0
        LDF     *+AR0(4),R0     ;M2.P_Time[PartB]
        CALL    _expon
;Schedule a Part departure after processing, considering the travel time
```

```
        ADDF    @CurPart+4,R0   ;Considering the travel time of robot
        STF     R0,@EveList3+3  ;Next departure time
        STF     R0,@CurPart+5   ;Current part finished time
        LDI     1,R4            ;Part type B = 1
        STI     R4,@EveList3+1  ;Next departure part type
        STI     R4,@CurPart
;Update Machine 2 busy period
        ADDF    @SumM2,R0
        SUBF    @CurPart+4,R0
        STF     R0,@SumM2
        B       LA8
*** B   LA8 ;BRANCH OCCURS
LA4     LDI     @PNumInQ,AR0    ;M2 is not idle
        LDI     *+AR0(1),R1
        ADDI    1,R1
        STI     R1,*+AR0(1)     ;Increase queue length
        LDI     @QLimit,R2      ;Check if input queue full
        CMPI    *+AR0(1),R2
        BGE     LA6
        LDI     3,R0
        CALL    _exit
LA6     LDI     @PInputQ,AR1    ;The first position of the queue
        LDI     *+AR0(1),R1     ;number in queues
        MPYI    6,R1
        ADDI    R1,AR1          ;InputQ[PartB][NumInQ[PartB]]
        LDI     @PInputQ,AR0
        LDI     5,RC            ;6 data to pass
        RPTB    LA7
        LDI     *AR0++,R1
LA7     STI     R1,*AR1++       ;InputQ[PartType][NumInQ[PartType]] = InputQ[PartType][0]
;If Part B arrive, schedule next Part B arrival
LA8     LDF     @MeArriB,R0     ;Schedule next Part B arrival
        CALL    _expon
        ADDF    @Time,R0
        STF     R0,@InputQ0+3
        STF     R0,@EveList1+3

        LDI     @PTotNum,AR0    ;Increase total entering part number
        LDI     1,R0
        ADDI    *+AR0(1),R0
        STI     R0,*+AR0(1)
        RETS

*---------------------------------------------------------------------------
* FUNCTION DEF : _arrivea
* schedule the next arrival event and handle current arrival event
* Input        : None
* Register     : R0, R1, R2, R4, AR0, AR1, RC, RS, RE
* Output       : None
*---------------------------------------------------------------------------
_arrivea:
        NOP
LAA0    LDI     *+AR6(2),R0
        CMPI    1,R0
        BNZ     LAA0            ;Check if new part arrive
;Read in arrived part
        LDI     @PM1_M2Part,AR0 ;Point to shared memory
        LDI     *AR0,R0         ;WIP queue 0 = M1_M2Par
        LDI     *+AR0(1),R1
        LDI     *+AR0(2),R2
        LDF     *+AR0(3),R3
        LDF     *+AR0(4),R4
        LDF     *+AR0(5),R5
        STI     R0,@WIPQ0
        STI     R1,@WIPQ0+1
        STI     R2,@WIPQ0+2
        STF     R3,@WIPQ0+3
        STF     R4,@WIPQ0+4
        STF     R5,@WIPQ0+5
        LDI     0,R0
        STI     R0,*+AR6(2)     ;Parts read
```

```
;Check to see if Mill 1 is idle
        LDI     @M2+1,R0
        BNZ     LB4             ;M2 is not idle

        LDI     1,R0            ;M2 is idle, set it to busy
        STI     R0,@M2+1
;Load part from queue to machine 2, WIP to M2 is 3->5, Input Queue to M2 is 4->5,
        LDI     5,R1            ;Destination
        LDI     3,R0            ;Starting
        CALL    _rerobot        ;Travel time return in R0
        LDF     @Time,R1
        STF     R1,@CurPart+3   ;Entering time
        ADDF    R0,R1
        STF     R1,@CurPart+4   ;Start processing time
        LDI     @PM2,AR0
        LDF     *+AR0(3),R0     ;M2.P_Time[PartA]
        CALL    _expon
;Schedule a Part departure after processing, considering the travel time
        ADDF    @CurPart+4,R0   ;Considering the travel time of robot
        STF     R0,@EveList3+3  ;Next departure time
        STF     R0,@CurPart+5   ;Current part finished time
        LDI     0,R4            ;Part type A = 0
        STI     R4,@EveList3+1  ;Next departure part type
        STI     R4,@CurPart
;Update Machine 2 busy period
        ADDF    @SumM2,R0
        SUBF    @CurPart+4,R0
        STF     R0,@SumM2
        B       LB8
*** B   LA8 ;BRANCH OCCURS
LB4     LDI     @PNumInQ,AR0    ;M2 is not idle
        LDI     *AR0,R1
        ADDI    1,R1
        STI     R1,*AR0         ;Increase queue length
        LDI     @WIPLimit,R2    ;Check if input queue full
        CMPI    *AR0,R2
        BGT     LB6
        LDI     1,R1
        STI     R1,*+AR6(1)     ;WIP queue full
        LDF     @EveList3+3,R0  ;Next Departure time
        STF     R0,*+AR6(4)     ;Inform M1 next part departure time till block released

LB6     LDI     @PWIPQ,AR1      ;The first position of the queue
        LDI     *AR0,R1         ;number in queues
        MPYI    6,R1
        ADDI    R1,AR1          ;InputQ[PartB][NumInQ[PartB]]
        LDI     @PWIPQ,AR0
        LDI     5,RC            ;6 data to pass
        RPTB    LB7
        LDI     *AR0++,R1
LB7     STI     R1,*AR1++       ;InputQ[PartType][NumInQ[PartType]] = InputQ[PartType][0]
;Schedule next Part A arrival to be infinite
LB8     LDI     1,R1
        STI     R1,*AR6         ;Indicate the event has been processed

        LDF     @BigNum+1,R0
        STF     R0,@EveList2+3  ;Reset the part A arrival event
        LDI     @PTotNum,AR0    ;Increase total entering part number
        LDI     1,R0
        ADDI    *AR0,R0
        STI     R0,*AR0
        RETS

*---------------------------------------------------------------------------
* FUNCTION DEF : _depart
* handle current departure event
* Input        : None
* Register     : R0, R1, R2, R3, R4, R5, AR0, AR1, RC, RS, RE, DP
* Output       : None
*---------------------------------------------------------------------------
```

```
_depart:
        LDI     @EveList3+1,R4  ;Current part type in R4
;Update current part stay in system, it is realy partial of part stay, need to plus
;the stay time at machine 1
        LDF     @CurPart+5,R0
        SUBF    @CurPart+3,R0
        LDI     @PSumStay,AR0
        ADDI3   R4,AR0,AR1      ;
        ADDF    *AR1,R0
        STF     R0,*AR1         ;SumStay[PartType] += CurPart.P_Finish - CurPart.P_Enter
        LDI     @PSumPart,AR0   ;Update finished parts
        ADDI3   R4,AR0,AR1
        LDI     1,R0
        ADDI3   R0,*AR1,R1
        STI     R1,*AR1
;Check to see if all queues are empty
        LDI     @NumInQ,R1
        BNZ     LD1             ;WIP queue is not empty
        LDI     @NumInQ+1,R2
        BNZ     LD1             ;Input queue is not empty
        LDI     0,R3            ;All queues are empty
        STI     R3,@M2+1        ;Set Machine 2 idle
        STI     R3,*+AR6(1)     ;M2 queues are not full
        STI     R4,@EveList3+1  ;Part type does not matter
        LDF     @BigNum+1,R3
        STF     R3,@EveList3+3  ;Set next departure event infinite
        B       LDE
*** B   LDE  ;BRANCH OCCURS
;Queues are not empty, decide next part to be processed according to the dispatching rule.
;Here FIFO is adopted
LD1     LDI     @NumInQ,R1
        BNZ     LD2
        LDI     1,R4            ;If WIP queue is empty, next part is B
        B       LD5
LD2     LDI     @NumInQ+1,R2
        BNZ     LD3
        LDI     0,R4            ;If Input queue is empty, next part is A
        B       LD5
LD3     LDF     @WIPQ0+9,R3     ;If both are not empty, FIFO
        CMPF    @InputQ0+9,R3   ;if WIPQ[1].P_Enter <= INPUTQ[1].P_Enter
        BGT     LD4             ;If part B arrival first
        LDI     0,R4            ;Part A arrive first
        B       LD5
LD4     LDI     1,R4            ;Part B arrive first
LD5     LDI     1,R0
        STI     R0,@M2+1        ;Set Machine 2 busy
        LDI     @PNumInQ,AR1
        ADDI    R4,AR1
        SUBI3   R0,*AR1,R2      ;Reduce queue length
        STI     R2,*AR1
;Reset WIP queue is not full
        LDI     @NumInQ,R1      ;M2QFull = (NumInQ[0]<WIPLimit)?0:1
        CMPI    @WIPLimit,R1
        LDILT   0,R2
        LDIGE   1,R2
        STI     R2,*+AR6(1)     ;M2 WIP Queue not full, M2QFULL

;Update delay accumulator
        LDI     42,R3
        MPYI3   R3,R4,AR0       ;Index position
        ADDI    @PWIPQ,AR0      ;Head of queues
        LDI     6,R0
        ADDI3   R0,AR0,AR1      ;First position of queues
        LDI     3,R0
        ADDI3   R0,AR1,AR0      ;First Part in queue Entering time
        LDF     @Time,R5        ;Current simulation clock
        SUBF    *AR0,R5         ;Delay time = Time - P_Enter
        LDI     @PSumDelay,AR0
        ADDI    R4,AR0
        ADDF3   R5,*AR0,R3      ;Sum of delay
        STF     R3,*AR0
```

```
;Current Part
        LDI     @PCurPart,AR0   ;Current part pointer in AR0, AR1 points to the first in queue
        LDI     5,RC
        RPTB    LD6
        LDI     *AR1++,R3
LD6     STI     R3,*AR0++       ;CurPart = InputQ[PartType][1]
;Maintain the queue
        LDI     42,R3
        MPYI3   R3,R4,AR0       ;Index position
        ADDI    @PWIPQ,AR0      ;Head of queues
        LDI     6,R0
        ADDI    R0,AR0          ;First in queues
        ADDI3   R0,AR0,AR1      ;Next in queue

        LDI     @PNumInQ,AR2    ;Point to the NumInQ
        ADDI    R4,AR2
        LDI     *AR2,R0
        ADDI    1,R0            ;Fill the last position with zeros
        MPYI    6,R0            ;The total number need to be relocated
        SUBI    1,R0
        LDI     R0,RC
        RPTB    LD7
        LDI     *AR1++,R2
LD7     STI     R2,*AR0++
;Schedule next departure
        LDI     6,R0
        LDI     5,R1            ;Parts leave machine 2 and exit the FMS
        CALL    _rerobot        ;Request robot
        ADDF    @Time,R0
        STF     R0,@CurPart+4   ;Starting time
        LDI     @PM2,AR0
        ADDI3   R4,AR0,AR1
        LDF     *+AR1(3),R0     ;Processing time
        CALL    _expon
        ADDF    @CurPart+4,R0
        STF     R0,@EveList3+3  ;Schedule next departure
        STI     R4,@EveList3+1  ;Event related part type
        STF     R0,@CurPart+5   ;Current part finished time
        STF     R0,*+AR6(4)     ;Inform M1 next part departure time till block released, M2D_Time
;Update Machine busy period
        ADDF    @SumM2,R0
        SUBF    @CurPart+4,R0
        STF     R0,@SumM2
LDE     RETS

*---------------------------------------------------------------------------
* FUNCTION DEF : _rerobot
* handle robot request, return robot travel time.
* Input        : R0,initial position, R1, destination position
* Register     : R0, R1, AR0, IR0, IR1, (only AR0,IR0,IR1 restored)
* Output       : R0 travel time
*---------------------------------------------------------------------------
_rerobot:
;Calculate travel time, robot need to travel from current position to initial position,
;then from initial position to the destination
        PUSH    AR0
        PUSH    AR1
        PUSH    IR0
        PUSH    IR1

        LDI     @PGrephon, AR1          ;Pointer to the robot
        LDI     *+AR1(1),IR0
        MPYI    8,IR0
        ADDI    R0,IR0                  ;From current position to starting position
        LDI     @PRNetwork,AR0
        LDI     R0,IR1
        MPYI    8,IR1
        ADDI    R1,IR1                  ;From starting position to destination
;Robot traveltime = RNetwork[Grephon.R_Pos][initpos] + RNetwork[initpos][destpos]
        ADDF3   *+AR0(IR1),*+AR0(IR0),R0
        STI     R1,*+AR1(1)             ;Next robot position
```

```
        LDF     @Time,R1
        CMPF    *+AR1(3),R1
        BGE     LR1
        ADDF    *+AR1(3),R0             ;Robot is occupied, considering some waiting time
        SUBF    R1,R0
;Next time robot stay at initial position and available time is current time plus travel time
LR1     ADDF    R0,R1                   ;Next available time
        STF     R1,*+AR1(3)
        LDF     2.0,R1
        MPYF    R0,R1
        ADDF    *+AR1(4),R1             ;Update robot utilization
        STF     R1,*+AR1(4)

        POP     IR1
        POP     IR0
        POP     AR1
        POP     AR0

        RETS
*-------------------------------------------------------------------------
* FUNCTION DEF : _report
* report simulation results
* Input        : None
* Register     : R0
* Output       : None
*-------------------------------------------------------------------------
_report:

        LDF     @EndTime,R0
        CALL    INV_F30
        RND     R0,R5           ;R5=1/Time

        LDF     @SumM2,R0
        MPYF    R5,R0
        STF     R0,@AveM2       ;Utilization of M1

        LDI     @PGrephon,AR0
        LDF     *+AR0(4),R0
        MPYF    R5,R0
        STF     R0,@AveR        ;Utilization of robot

        LDI     0,R4
LRE1    LDI     @PSumInQ,AR0
        ADDI    R4,AR0
        LDF     *AR0,R0
        MPYF    R5,R0
        LDI     @PAveInQ,AR0
        ADDI    R4,AR0
        STF     R0,*AR0         ;Average in queues

        LDI     @PSumPart,AR0
        ADDI    R4,AR0
        FLOAT   *AR0,R0
        CALL    INV_F30
        RND     R0,R2           ;R2 = 1/SumPart[i]

        LDI     @PSumDelay,AR0
        ADDI    R4,AR0
        LDF     *AR0,R0
        MPYF    R2,R0
        LDI     @PAveDelay,AR0
        ADDI    R4,AR0
        STF     R0,*AR0         ;Average delay

        LDI     @PSumStay,AR0
        ADDI    R4,AR0
        LDF     *AR0,R0
        MPYF    R2,R0
        LDI     @PAveStay,AR0
        ADDI    R4,AR0
        STF     R0,*AR0         ;Average stay
```

```
        ADDI    1,R4
        CMPI    2,R4
        BLT     LRE1
        RETS

*-----------------------------------------------------------------------
* FUNCTION DEF : _barrier
* Unusually exit
* Input        : none
* Register     : R0
* Output       : none
*-----------------------------------------------------------------------
_barrier:
        LDP     SharedMem1
        LDI     @Barrier1,R0
        ADDI    1,R0
        STI     R0,@Barrier1
B1      LDI     @Barrier1,R0
        CMPI    2,R0
        BLT     B1
        LDP     @_main
        RETS

*-----------------------------------------------------------------------
* FUNCTION DEF : _exit
* Unusually exit
* Input        : none
* Register     : none
* Output       : none
*-----------------------------------------------------------------------
_exit:
        LDP     @_main
        STI     R0,@SysExit
EXITSIM B       $

*-----------------------------------------------------------------------
* FUNCTION DEF : _expon
* Generate a random number with exponential distribution
* Input        : R0 is the mean
* Register     : R1
* Output       : R0 is the random number with exponential distribution
*-----------------------------------------------------------------------
_expon:
*Fast 32 bit uniform random number generator
        PUSH    R1
        PUSHF   R1

        NEGF    R0, R1          ; R1 is the negative of the mean
        PUSH    R1
        PUSHF   R1              ; Save -mean in the stack, all 40 bits
        LDI     @SEED,R0        ; Call here for last SEED
        MPYI    @A,R0           ; Calculate r = (A*r+C)%m
        ADDI    @C,R0           ;
        AND     @MASK1, R0      ; m=2^23
        STI     R0,@SEED        ; Result is returned
        ADDI    1,R0            ;
        FLOAT   R0,R0           ;
        MPYF    @INVE23,R0      ; r/m
        CALL    _ln             ; ln(u)
        POPF    R1              ; -Mean
        POP     R1
        MPYF    R1, R0          ; random number =  -mean*ln(u)

        POPF    R1
        POP     R1
        RETS


*-----------------------------------------------------------------------
*Define references
```

```
*---------------------------------------------------------------------------
*Define the entry point of subroutines
SUB     .word _main
        .word _arriveb
        .word _arrivea       ;parts arrival invoke same subroutine
        .word _depart
PSUB    .word SUB
STACK   .word  $+1          ; Bottom of the Stack
        .end
```

Note:

1. Subroutines _ln and _INVF30 are the same as in 1MhardL1.asm
2. Only one set of program is listed here

C.3.3 Sample Data Recorded from the Prototype Simulator

Table A9 and A10 show two set of data recorded for simulating 4-workstation case. There are total 15 sets of such data generated by randomly changing seeds.

A9 Experiment No.1 for simulating 4-workstaton (seeds 0/0)

Experiment Data for 4-machine Simulation **Date: Apr. 5** **Experiment No. 1**					
Execution Cycles		Master 1 816294		Master 2 817472	
Simulation Time		14400		14404	
Seeds		0	0	0	0
Parameters		**DSP 1** **Lathe 1**	**DSP 2** **Mill 1**	**DSP 3** **Lathe 2**	**DSP 4** **Mill 2**
MTBA	Part A	15	N/A	15	N/A
	Part B	N/A	20	N/A	20
No. Part Arrivals	Part A	987	948	944	973
	Part B	N/A	732	N/A	750
No. Part Processed	Part A	948	943	974	970
	Part B	N/A	723	N/A	747
No. Part Routed	Part A	32		32	
Avg in Queue	WIP	N/A	PA 1.633	N/A	PA 2.009
	Input	PA 1.172	PB 1.204	PA 1.035	PB 1.650
Util of Machine		0.409	0.702	0.403	0.732
Machine Blocked		0.070	N/A	0.089	N/A
Util of Robots		R1 0.374		R2 0.392	
Avg. stay on Mach.	Part A	25.216	33.195	25.092	38.520
	Part B	N/A	28.555	N/A	36.557
Avg. delay in Que	WIP	N/A	PA 24.625	N/A	PA 29.713
	Input	PA 16.206	PB 23.730	PA 16.612	PB 31.679

A10 Experiment No.15 for simulating 4-workstaton (seeds 702749/431012)

Experiment Data for 4-machine Simulation					
Date: Apr. 5 **Experiment No. 15**					
Execution Cycles		Master 1 744417		Master 2 745057	
Simulation Time		14403		14405	
Seeds		702749	431012	702749	431012
Parameters		**DSP 1 Lathe 1**	**DSP 2 Mill 1**	**DSP 3 Lathe 2**	**DSP 4 Mill 2**
MTBA	Part A	15	N/A	15	N/A
	Part B	N/A	20	N/A	20
No. Part Arrivals	Part A	990	954	985	1018
	Part B	N/A	701	N/A	763
No. Part Processed	Part A	954	951	1018	1017
	Part B	N/A	699	N/A	761
No. Part Routed	Part A	34		34	
Avg in Queue	WIP	N/A	PA 2.237	N/A	PA 2.124
	Input	PA 1.321	PB 1.695	PA 1.972	PB 1.737
Util of Machine		0.399	0.746	0.421	0.740
Machine Blocked		0.114	N/A	0.131	N/A
Util of Robots		R1 0.375		R2 0.405	
Avg. stay on Mach.	Part A	28.387	43.239	37.747	38.617
	Part B	N/A	39.714	N/A	37.640
Avg. delay in Que	WIP	N/A	PA 33.862	N/A	PA 30.072
	Input	PA 19.054	PB 34.899	PA 28.729	PB 32.870

Table A11 and A12 show two set of data recorded for simulating 2-workstation case. There are total 9 sets of such data generated by changing arrival pattern.

A11 Experiment No.1 for simulating 4-workstaton (arrival 15/15)

Experiment Data for 4-machine Simulation **Date: Apr. 6** **Experiment No. 1**					
Execution Cycles		Master 1 932405		Master 2 932117	
Simulation Time		14414		14401	
Seeds		0	0	0	0
Parameters		**DSP 1** **Lathe 1**	**DSP 2** **Mill 1**	**DSP 3** **Lathe 2**	**DSP 4** **Mill 2**
MTBA	Part A	15	N/A	15	N/A
	Part B	N/A	15	N/A	15
No. Part Arrivals	Part A	991	973	982	996
	Part B	N/A	973	N/A	987
No. Part Processed	Part A	974	972	996	992
	Part B	N/A	972	N/A	985
No. Part Routed	Part A	16		16	
Avg in Queue	WIP	N/A	PA 2.400	N/A	PA 2.942
	Input	PA 1.692	PB 2.552	PA 2.108	PB 3.189
Util of Machine		0.420	0.794	0.425	0.808
Machine Blocked		0.119	N/A	0.202	N/A
Util of Robots		R1 0.408		R2 0.416	
Avg. stay on Mach.	Part A	34.483	44.387	40.884	51.260
	Part B	N/A	42.545	N/A	51.535
Avg. delay in Que	WIP	N/A	PA 35.550	N/A	PA 42.634
	Input	PA 24.876	PB 37.810	PA 30.619	PB 46.624

A12 Experiment No.9 for simulating 4-workstaton (arrival 25/25)

Experiment Data for 4-machine Simulation					
Date: Apr. 6 **Experiment No. 9**					
Execution Cycles		Master 1 443328		Master 2 441835	
Simulation Time		14424		14425	
Seeds		0	0	0	0
Parameters		**DSP 1** **Lathe 1**	**DSP 2** **Mill 1**	**DSP 3** **Lathe 2**	**DSP 4** **Mill 2**
MTBA	Part A	25	N/A	25	N/A
	Part B	N/A	25	N/A	25
No. Part Arrivals	Part A	571	559	587	595
	Part B	N/A	580	N/A	584
No. Part Processed	Part A	559	559	595	595
	Part B	N/A	578	N/A	583
No. Part Routed	Part A	10		10	
Avg in Queue	WIP	N/A	PA 0.376	N/A	PA 0.373
	Input	PA 0.187	PB 0.359	PA 0.156	PB 0.389
Util of Machine		0.226	0.250	0.253	0.502
Machine Blocked		0.003	N/A	0.002	N/A
Util of Robots		R1 0.249		R2 0.262	
Avg. stay on Mach.	Part A	12.468	18.940	10.974	18.080
	Part B	N/A	13.825	N/A	14.687
Avg. delay in Que	WIP	N/A	PA 9.692	N/A	PA 9.036
	Input	PA 5.092	PB 8.960	PA 3.523	PB 9.599

Vita

Dong Xu

Dong Xu was born in Sichuan Province, the People's Republic of China in 1971. He received his bachelor of Science in Automation Control from Tsinghua University in July 1993. In 1996, He graduated from Tsinghua University with his Master of Science in Systems Engineering. After that, he came to the United States of American to purse a Ph.D. degree. During the first three years, he studied in the University of Toledo, Toledo, OH. He received his Master of Science in Industrial Engineering there before he transferred to Virginia Polytechnic Institute and State University to continue his Ph.D. study. He completed his Ph.D. in Industrial and Systems Engineering with a concentration in Manufacturing Systems in July 2001 from Virginia Polytechnic Institute and State University. Currently, he is working for Microsoft as a Software Design Engineer in Test.