Reconfigurable Hardware-Based Simulation Modeling of Flexible Manufacturing Systems

Wei Tang

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

Doctor of Philosophy

In

Industrial & Systems Engineering

Nov 18, 2005

Blacksburg, Virginia

Keywords: reconfigurable, hardware, simulator, simulation, FPGA, FMS

Reconfigurable Hardware-Based Simulation Modeling of Flexible Manufacturing Systems

Wei Tang

(ABSTRACT)

This dissertation research explores a reconfigurable hardware-based parallel simulation mechanism that can dramatically improve the speed of simulating the operations of flexible manufacturing systems (FMS). Here reconfigurable hardware-based simulation refers to running simulation on a reconfigurable hardware platform, realized by Field Programmable Gate Array (FPGA). The hardware model, also called simulator, is specifically designed for mimicking a small desktop FMS. It is composed of several micro-emulators, which are capable of mimicking operations of equipment in FMS, such as machine centers, transporters, and load/unload stations.

To design possible architectures for the simulator, a mapping technology is applied using the physical layout information of an FMS. Under such a mapping method, the simulation model is decomposed into a cluster of micro emulators on the board where each machine center is represented by one micro emulator. To exploit the advantage of massive parallelism, a kind of star network architecture is proposed, with the robot sitting at the center. As a pilot effort, a prototype simulator has been successfully built.

A new simulation modeling technology named *synchronous real-time simulation* (SRS) is proposed. Instead of running conventional programs on a microprocessor, this new technology adopts several concepts from electronic area, such as using electronic signals to mimic the behavior of entities and using specifically designed circuits to mimic system resources. Besides, a time-scaling simulation method is employed. The method uses an on-board global clock to synchronize all activities performed on different emulators, and by this way tremendous overhead on synchronization can be avoided. Experiments on the prototype simulator demonstrate the validity of the new modeling technology, and also show that tremendous speedup compared to conventional software-based simulation methods can be achieved.

ACKNOWLEDGMENTS

While completing this exciting research, which I deem as the most significant achievement in my life so far, I would like to express my great appreciation to several persons. Without their help, support, and encouragement, I would never have been able to finish this work.

First of all, I would like to thank my major advisor Dr. F. Frank Chen, for his inspiring and encouraging way to guide me to a deeper understanding of knowledge, and his invaluable comments during the course of this dissertation research work. This research involves an innovative technical advancement, which requires considerable investments and bears inherent uncertainty. I feel so lucky that I always receive firm support from Dr. Chen all the way to completion. The generous multiple-year financial support via a grant from the National Science Foundation (DMI-9996417) awarded to Dr. Chen and the continuing financial support from my final year of this study from the Center for High Performance Manufacturing at Virginia Tech are gratefully acknowledged.

I would also like to express a special thank to Dr. Dong Xu, a former member of our research group, for a fruitful collaboration. Some of my thoughts were directly inspired by his previous research work. Dr. Xu’s work provides an excellent foundation which I used to further advance the state of the art of hardware-based simulation technology.

Many thanks go to my committee members: Dr. Michael P. Deisenroth, Dr. Subhash C. Sarin, Dr. C. Patrick Koelling and Dr. Sandeep Shukla. Their valuable comments and suggestions helped to keep this research on the right track. Many extensive discussions and warm debates about this research stimulated new ideas contributing to the development of this research.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 INTRODUCTION

1.1 Background

A flexible manufacturing system (FMS) is a group of numerically controlled machine tools with an automated material handling system and a central supervisory computer system. It is characterized by high automation and flexible alternatives in parts routing, tool delivery, capacity planning, and bottleneck detecting. The success of an FMS heavily depends on having effective planning, scheduling, and control. Though numerous alternatives provide the foundation for the required flexibility of FMS, it is extremely difficult for the analytical methods to study the system taking all these alternatives into account to generate 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 the flexible manufacturing systems. Traditionally, computer-based simulation has been used for capacity planning, bottlenecks detection, and creating and testing manufacturing schedules. However, in the real manufacturing environment, unexpected events such as machine failures, part delivery delays may happen and make the best plan developed by the best production planning methodology ineffective. Therefore, high-speed reactive scheduling or rescheduling tools are often needed to make modifications to the original schedule in order to reflect current shop floor environment. Recent efforts have been gathered towards using simulation for real-time rescheduling. The real-time approach to using simulation for FMS analysis requires the handling of large amount of knowledge to prepare the simulation model and to evaluate its outputs. When applying various software-based simulation methods to the real-time simulation, the speed of the simulations is far from enough in responding to the real-time requirements for a shop floor control. To realize full real-time simulation and facilitate the on-line control of the FMS, especially under the situations when unexpected events occur, this research explores a hardware-based, FMS specific simulation method, which takes advantage of implementing digital electronic technology directly into the simulation.

Previously a new type of hardware-based parallel simulator was proposed by researchers at the FMS Lab of Virginia Tech [Xu, et al. 1999]. The simulator is a multi-microprocessor based digital circuit board which consists of a collection of "micro emulator units" used to mimic the complete set of machinery and equipments in an FMS, such as machining centers, transporters, etc. Each micro emulator unit consists of a microprocessor with its supporting peripheral circuits for local memory and communication. The operating logics of the designated resource, such as processing times, arrival patterns for different parts, and down time of machines can be captured via programs running on each micro emulator unit. In this way, a printed circuit board (PCB) can be built to capture the total operating scenario of an FMS, and the completed PCB can therefore be viewed as a lithograph of a real flexible manufacturing system. The simulator also implemented some innovative design concepts such as system mapping technology, scaled real-time management and digital signal processing (DSP) functions. Results showed that significant speedup over traditional software simulation can be obtained. However, since this simulator was based on DSP technology, and all simulation procedures are realized by C or Assembly language, it still relies heavily on software execution, though with some hardware assistance. The simulation carried out on this simulator still cannot get rid of some major time-consuming activities of common software simulators, such as instruction fetching, decoding and memory accessing. Therefore, it is still limited in providing appropriate simulation speed for real-time applications.

With the rapid development of new reconfigurable devices, such as Field Programmable Gate Array (FPGA), it is possible to build a new simulation architecture which can boost simulation speed even further. Reconfigurable computing (RC) concept is introduced here as an innovative approach for system design, which tries to cope with the problem of inefficiency of conventional computing systems. It has dissolved the border between hardware and software and joined the potentials of both. The user designed FPGA chips can provide the simulator with more powerful computing recourses, thus significantly enhance the simulation speed. In this research, an innovative parallel reconfigurable simulator based on FPGA is proposed.

1.2 Motivations

1.2.1 Why real-time simulation-based scheduling?

Theoretically, “real time” should mean making a decision immediately. Practically speaking, the speed at which a decision is needed to be considered as real-time depends on system parameters such as the magnitude of part processing times and the flexibility of the system [Harmonosky and Robohn 1991]. In general, FMSs are more sensitive to system disturbances than conventional manufacturing systems because of tighter synchronization, system integration, and interdependencies among automated components. Hence, they require an immediate response to changes in system states, and this can be achieved by real-time scheduling, where decisions are based on actual system states, such as arrival of parts, machine states (up or down), queues at machines, tool breakages, rushed jobs, and many other system disturbances [Kim 1994]. A wide range of techniques, from discipline of control theory, operations research, and artificial intelligence, have been investigated as candidates for the construction of decision support tools [Rogers and Rosalee 1993]. Since for a long time simulation has been regarded as an effective tool in system design, evaluating the impact of changing system parameters, it is natural to attempt to use simulation for real-time decision making support.

1.2.2 Why exploring hardware simulation?

There exist some significant drawbacks to software-based system simulation. For example, building and validating a simulation model requires tremendous work and the model 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 highly desirable. On the other hand, hardware acceleration, usually based on a network of custom processors, can be used to increase the speed and capacity of simulation by a large magnitude [Xu 2001].

1.2.3 Why reconfigurable?

The motivation for a dynamically reconfigurable hardware-based parallel simulator is derived from several observations. First, it provides hardware functional design with some flexibility. Traditional hardware design is usually based on application specific integrated circuit (ASIC). After fabrication, the design cannot be changed anymore. Once there are errors found in the design or some new features need to be added, the cost could be very huge. Second, the configuration of advanced manufacturing systems may change from time to time according to changing production requirement, so the simulator itself also needs to be easily reconfigured in order to capture all the new features in the shop floor so as to achieve realistic simulation results. For traditional ASIC hardware design, after a function unit of the simulator is implemented, there is no simple way to make any change for tracking new features of the real system. On the other hand, field programmable logic devices are ideal to realize the post-manufacture design and implementation, and designs can be changed even in shop floor environment effortlessly. The third reason is for economical soundness. Traditional application specific hardware designs require extensive professional knowledge, long development cycle time and high cost of human & financial resources. Even at run time, statically configured architecture is inefficient because resources needed during one phase are often underutilized in other phases. All those drawbacks could be avoided with state-of-the-art circuit designs based on reconfigurable device technology. At application level, the re-configurability of the system allows a greater level of code reuse as well as the incorporation of new capabilities into applications directly at run-time. Therefore, the reconfigurable devices are deemed ideal for certain applications in real-time manufacturing simulation.

1.2.4 Why new simulation paradigm?

Since the simulation will run on a platform that is completely different from microprocessor system, traditional software simulation paradigm might not fit. The majority of simulation-based scheduling research is focused on parallel discrete event simulation (PDES) because of its capability of simulating dynamic and complex system. However, there are some inherent drawbacks related to PDES. One of the most difficult issues is related to synchronization, which is to assure each logic process to abide by local causality constraint. Although tremendous research efforts had been made and many solutions had been proposed to deal with this, the overhead of synchronization remains considerable high. Global clock synchronization may be used to bypass synchronization problem in time-stepped simulation, but the simulation speed is generally unacceptable on a general- purpose processor-based platform for any real-time applications. In this research, a new simulation paradigm using reconfigurable hardware platform will be proposed. In the new simulation paradigm all elements in traditional simulation are transferred into hardware domain.

1.3 Objectives

The objectives of this research can be summarized as follows:

- Summarize the pros and cons of previously proposed DSP based simulator. Identify the technology barriers and seek solutions to those problems. No matter how powerful a DSP processor is, it can only improve a limited portion of manufacturing simulation. Most of the simulation run are still based on microprocessor. Therefore, DSP-based simulator is not fundamentally different from traditional software simulation.
- Explore the basic concepts about the reconfigurable hardware simulation modeling technology. Because no prior attempts have been found in design of a manufacturing system simulator completely based on hardware platform, the immediate objective of this research is to explore the feasibility of the hardware-based parallel simulation. Before implementation of the hardware simulator, all designs will be simulated under a specific system integration environment. This will make sure the designs are logically verified.
- Develop a prototype simulator and a corresponding simulation mechanism on an FPGA platform. This research will first be carried out based on some off-the-shelf FPGA development kit, with corresponding function units developed by hardware description language (HDL). 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 significantly different from traditional system simulation methods.
- Experiment on the prototype simulator, collect and analyze data, and evaluate the performance of this hardware-based simulator. Also, the simulated results will be compared

to those from an identical system developed with a software simulation package, as well as the desktop automated manufacturing system located in FMS lab at Grado Department of ISE, Virginia Tech. This procedure will ensure that the hardware simulator is validated.

- Identify the limitations of hardware-based parallel simulator through this research. There are always positive and negative aspects regarding new technology and design. In this case, for example, with all the benefits brought by reconfigurable devices, there are also some potential shortcomings, such as immaturity of the new technology and shortage of supporting intellectual properties (IP).

A prototype reconfigurable hardware simulator will be built to mimic the laboratory FMS located in the Flexible Manufacturing Systems Laboratory at the Grado Department of Industrial and Systems Engineering, Virginia Polytechnic Institute & State University. Components of the laboratory FMS which will be considered in this research are the parts, workstations (machines, load/unload station), storage devices (index tables), transporting 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 when unexpected disruption arises in shop floor.

1.4 Contribution of this research

a. An innovate simulation modeling technology --- synchronous real-time simulation

Synchronous real-time simulation (SRS) is a special electronic signal driven simulation. The simulation tries to map activities occurring in real-world to a micro-world inside circuitry where electronic signals travel at lightening fast speed. Events, which cause the change of system status, are not explicitly represented in SRS. They are triggered when signals arrive or depart. All events are processed immediately after they take place, so event list is not needed in the simulation. State updating and event scheduling are realized by hardware connection in one clock cycle, and no computation is required to process events. Therefore, microprocessor is not necessary in SRS. Local processes are naturally synchronized by a global clock, avoiding tremendous overhead required on synchronization mechanisms in traditional PDES.

b. Introducing reconfigurable computing into manufacturing system simulation area

Computer based system simulation is realized through some computer language or software package, such as SIMAN, Pro-Model or Arena. These software-based approaches have many advantages such as flexible, versatile, graphic intensive. However, while dealing with time critical applications they do not seem to be capable enough. This research explores a methodology of building a powerful real-time simulator by transforming software simulation functionality into reconfigurable hardware realization, thus breaking the speed barrier typically inherent in software. By realizing the FPGA based hardware emulator for machines, the speed of simulator is uplifted significantly. With massive hardware parallelism being employed, the speed barrier problem caused by scaling the time of intensive computation is alleviated. Meanwhile the resource limitation of general-purpose microprocessor is no longer an issue. Due to the limitation of DSP microprocessor, the former work [Xu 2001] only addresses a two-machine model based on time-scaled hardware simulation, while its global clock control mechanism is not yet mature. This research does not use the clock-stopping method proposed before, which is cumbersome to realize. Instead, high performance FPGA chips are employed.

c. Building a hardware simulator prototype

The proposed simulator is realized on an up-to-date FPGA based platform. Some unique technologies are implemented into the design, namely innovate simulation engine, time-scaling mechanism, special star network architecture and reconfigurable system integration. In this research a hardware simulation model of a small FMS consisting of four machine centers has been built. Extensive experiments have been carried out. Results showed that the hardware simulator can achieve results identical to those from its software counterpart, but with a much faster execution speed. A solid ground has been laid for further research of building simulator based on more complex systems.

1.5 Research Plan

1. At the beginning, literature review and theoretical justification are finished. The previous DSP-based simulator prototype is examined and new design methodologies are proposed. Preliminary implementation has been done on the development platform. A system with RISC processor, dedicated bus, timer and interrupt controller and memories has been constructed. A program has been written to realize the event-driven simulation. A full functional hardware simulator based on 2-machine cell is designed and verified.

2. At the second stage of this research, a four-machine center simulator has been designed and implemented. System verification has been carried out based on circuit simulation results. Comparison between the results from hardware simulation and those from software simulation clearly showed the validity of this new simulator.

3. At the final stage, a full functional hardware-based simulator has been implemented on FPGA platform. This simulator is able to simulate the desktop FMS, which includes four machine centers, two robots, two conveyers, and pallets, inspection station, etc. The simulator has been built on a Virtex-4 based MicroBlaze development kit. The performance of the hardware-based simulator has been thoroughly evaluated, and the hardware model has been validated.

1.6 Organization

This dissertation is organized as follows. In Chapter 2, a brief introduction to flexible manufacturing system is given, and various simulation-based scheduling applications in manufacturing system are introduced. The objective of building a hardware simulator for FMS real-time scheduling is described in detail. Chapter 3 introduces various methodologies of accelerating PDES, and describes a hardware-based simulator previously implemented at Virginia Tech FMS lab. In the final part, an innovative simulation modeling technology based on reconfigurable device is proposed. Chapter 4 describes details of methodology and design of the hardware-based re-configurable simulator, including the mapping technique, possible system architectures, and time-scaling 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 simulator is introduced. Details about building a FPGA based circuit, simulating the design, and other related issues for implementation are discussed. In Chapter 6, experiments of simulation scenarios for 1-machine unit, 2-machine cell and 4-machine manufacturing system are designed and conducted in a logical development environment. Design of the simulator is implemented on a FPGA platform, and simulation results are colleted and compared with those from an identical software model. The last chapter, Chapter 7 summarizes conclusions and contributions of this research, and identifies some future research directions.

CHAPTER 2 REAL-TIME SIMULATION-BASED SCHEDULING FOR FMS

2.1 Flexible Manufacturing System

2.1.1 Introduction to FMS

An FMS is defined as a manufacturing system consisting of automatically reprogrammable machines (material processors), automated tool deliveries and changes, automated material handling and transport, and coordinated shop floor control [Askin and Standridge 1993]. An FMS is capable of processing different types of products in different sequence with short setup delays between operations. Typically, an FMS can produce over 20 different part types simultaneously in the system. Some large ones can produce over a hundred part types if a proper tool management system is provided. An FMS is distinguished from other types of manufacturing systems by following characteristics:

<u>High degree of functional integration</u>: The integration of an FMS can be classified in two aspects: integration among machining centers and integration between machines and the transportation system. The machining centers used in an FMS are usually very versatile and capable of performing different manufacturing operations with different specifications. These machining centers can directly access to a large set of tools from its tool magazine in order to meet different tooling requirements. The links between machines and transportation subsystem are also highly integrated by adopting automated material handling equipments, such as robots, and AGVs.

<u>Various operational sequences with short setup time</u>: When several machining centers are capable of performing the same kind of operation, there exist alternative paths to route parts. 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. Sometimes decision has to be made to choose one of the available paths according to some criteria. When system becomes complicated, it is very difficult for analytical models to analyze the performance of an FMS. Therefore, heuristics and simulation methods are widely used to assist the decision making.

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 work-pieces between the local tool magazines and the central tool storage. Tools have to be exchanged when part types to be processed are changed. Ideally, tool exchange should not interrupt the running process. If it is true that the tool exchange is not significant, it can be ignored in simulation models.

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. 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, which are usually not so important in other conventional production systems, are indispensable requirements for the operation of an FMS.

All these characteristics, while providing FMS with flexibility, are also making it very difficult to be formulated in analytical models. Thus, simulation is widely employed as a suitable tool to design, plan, schedule and evaluate an FMS, especially to provide answers to “what-if” questions.

2.1.2 Re-configurable manufacturing systems (RMS)

Although accepted to some extent in Japan and Europe, the overall FMS commercialization is not very successful, especially in the United States. The most widely claimed reasons for this include: low return on investment, complexity, lack of software reliability and needs for highly skilled workers. In term of design, FMS feature a hardware/software integral architecture meaning that it is difficult to identify the boundaries between the components and their functionalities, this may lead to relatively high supporting costs[Mehrabi et al. 2002].

The Engineering Research Center of the University of Michigan, Ann Arbor, along with more than 30 industrial partners, is working to develop a new generation of manufacturing systems that will be responsive to customer demands and changes to product mix. The Reconfigurable Machining System (RMS) is designed with rapid and constant changes in mind. A RMS is one designed at the outset for rapid change in its structure, as well as its hardware and software components, in order to quickly adjust its production capacity and functionality in response to sudden, unpredictable market changes as well as introduction of new products or new process technology [Koren and Ulsoy 1997]. Main features of RMS include:

- Modularity: In a reconfigurable manufacturing system, all major components are modular (e.g., structural elements, axes, controls, software, and tooling). When necessary, the components can be replaced or upgraded to better suit new applications. Modules are easier to maintain, thereby lowering life-cycle costs over current systems.
- Integratability: To aid in designing reconfigurable systems, a set of system configuration and integration rules must be developed. These rules should allow designers to relate clusters of part features and their corresponding machining operations to machine modules, thereby enabling product-process integration. In addition, the machine controls and the processing units must be designed for integration into a system.
- Diagnosability: Diagnosability has two aspects: detecting machine failure and unacceptable part quality.
- Customization: This characteristic drastically distinguishes RMS from both FMS and dedicated machine lines and allows a reduction in system and machine cost. It enables the design of a system for the production of a part family.

RMS is viewed as a promising technology. It has inherent capabilities for capacity adjustment, product variety and shorter changeover time. However, it still requires additional research and development in certain key technologies.

2.2 Shop floor scheduling of manufacturing systems

Manufacturing scheduling is a difficult problem, particularly when it takes place in an open, dynamic environment. In a manufacturing system, things rarely go as expected. Environment often changes dynamically. The system may be asked to do additional tasks that were not anticipated, and sometimes is allowed to omit certain tasks. The resources available to perform tasks are subject to change. Certain resources may become unavailable, and additional resources may be added. The beginning time and the processing time of a task are also subject to change. A task can take more or less time than anticipated, and tasks can arrive early or late. Because of its highly combinatorial aspects, dynamic nature and practical interest for manufacturing systems, the scheduling problem has been widely studied.

2.2.1 Basics of manufacturing system scheduling

Scheduling is one of the most important functions in a manufacturing company. It is the allocation of available production resources over time to meet some set of performance criteria. It basically answers two questions: which resources will be allocated to perform each task and when will each task be performed. Ideally, the objective function should consist of all costs in the system. In practice, however, such costs are often difficult to measure. Three types of decision-making goals seem to be prevalent in scheduling: efficient utilization of resources, rapid response to demands, and close conformance to prescribed deadlines. Typically the scheduling problem involves a set of jobs to be completed, where each job comprises a set of operations to be performed. Operations requiring machines and material resources must be performed according to some feasible technological sequence. Schedules are influenced by such factors as job priorities, due date requirements, release dates, cost restrictions, production levels, lot-size restrictions, machine availabilities, machine capabilities, operation precedence, resource requirements, and resource availabilities. Performance criteria typically involve trade-offs between holding inventory for the task, frequent production changeovers, satisfaction of production-level requirements, and satisfaction of due dates.

Except in very special cases, there are no good (polynomial-bounded) algorithms for scheduling problems, despite decades of intense research activity. In other words, it is in the non-deterministic polynomial (NP) complete domain, where satisfactory mathematical analysis and solutions are hard to find [Gonzalez and Sahni 1978]. Apart from the inherent complexity of the scheduling task, there are difficulties associated with the definition of the problem. The quality of a schedule is a function of several measures, some of which are easy to quantify, but others may be not properly defined and hence hard to measure, such as customer satisfaction or schedule robustness.

2.2.2 Scheduling methodologies

Over the years, the behavior and performance of job shops have been the focus of in the Operations Research (OR) literature. The most popular one of these topics is production scheduling (often referred to as job shop scheduling). Job shop scheduling can be thought of as the allocation of resources over a specified time to perform a predetermined collection of tasks. Job shop scheduling has received this large amount of attention, because it has the potential to dramatically decrease costs and increase throughput [Jones and Rabelo 1998]. Commonly used scheduling techniques include:

- Mathematical techniques

Mathematical programming has been applied extensively to job shop scheduling problems. Problems have been formulated using integer programming [Balas 1965; Balas 1967], mixed-integer programming [Balas 1969; Balas 1970], and dynamic programming [Srinivasan 1971]. Until recently, the use of these approaches has been limited because scheduling problems are NP-complete. To overcome these deficiencies, researchers began to decompose the scheduling problem into a number of sub-problems, proposing various techniques to solve them. In addition, new solution techniques, more powerful heuristics, and the computational power of modern computers have enabled these approaches to be used on larger problems. Still, difficulties in the formulation of material flow constraints as mathematical inequalities and the development of generalized software solutions have limited the use of these approaches.

- Dispatching rules

A common way of scheduling jobs dynamically in a manufacturing system is by means of dispatching rules. Dispatching rules are procedures designed to provide good solutions to complex problems in real-time. Dispatching rules can be have been classified mainly into three types according to the performance criteria [Wu 1987]. Type one uses simple priority rules, which are based on information related to the jobs, such as SPT, EDD, and FIFO. Type two consists of combinations of rules from type one. In this case the rules can depend on the situation that exists on the shop floor. A typical example is: SPT until the queue length exceeds 5, then switch to FIFO. Type three contains rules that are commonly referred to as Weight Priority Indexes. The idea here is to use multiple criteria simultaneously to determine the schedule. Target measurements are assigned weights to reflect their relative importance. Usually, an objective function f(x) is defined. The performance of a large number of these rules has been studied extensively using simulation techniques [Montazer and Wassenhove 1990]. These studies have been aimed at answering the question: If you want to optimize a particular performance criterion, which rule should you choose?

- Artificial intelligence (AI) techniques

Starting in the early 1980s, a series of new technologies were applied to job shop scheduling problems. They are generally referred as artificial intelligence (AI) techniques and include expert systems, knowledge-based systems, and several search techniques. Expert and knowledge-based systems were quite popular in the early and mid 1980s. They have four main advantages. First, they use both quantitative and qualitative knowledge in the decision-making process. Second, they are capable of generating heuristics that are significantly more complex than the simple dispatching rules described above. Third, the selection of the best heuristic can be based on information about the entire job shop including the current jobs, expected new jobs, and the current status of resources, material transporters, inventory, and personnel. Fourth, they capture complex relationships in elegant new data structures and contain special techniques for powerful manipulation of the information in these data structures. However, there are also some serious disadvantages. They can be time consuming to build and verify, as well as difficult to maintain and change. Moreover, since they generate only feasible solutions, it is rarely possible to tell how close that solution is to the optimal solution. Finally, since they are tied directly to the system they were built to manage, there is no such thing as a generic AI system.

- Artificial neural networks

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurones. This is true of ANNs as well. Rabelo [1990] was the first to use back-propagation neural nets to solve job shop scheduling problems with several job types, exhibiting different arrival patterns, process plans, precedence sequences and batch sizes. In that research the neural networks were tested in numerous problems and their performance (in terms of minimizing Mean Tardiness) was always better than each single dispatching rule.

- Neighborhood search methods

Neighborhood search methods provide good solutions and offer possibilities to be enhanced when combined with other heuristics. These methods iteratively add small changes to an initial schedule, which is obtained by any heuristic. Like hill climbing, these techniques continue to perturb and evaluate schedules until there is no improvement in the objective function. When this happens, the procedure is ended. Popular techniques that belong to this family include Tabu search, simulated annealing, and genetic algorithms. Each of these has its own perturbation methods, stopping rules, and methods for avoiding local optimum. The basic idea of Tabu search [Glover 1989; Glover 1990] is to explore the search space of all feasible scheduling solutions by a sequence of moves. A move from one schedule to another schedule is made by evaluating all candidates and choosing the best available, just like gradient-based techniques. Some moves are classified as Tabu (i.e., they are forbidden) because they either trap the search at a local optimum, or they lead to cycling. These moves are put onto something called the Tabu List, which is built up from the history of moves used during the search. These tabu moves force exploration of the search space until the old solution area is left behind. Simulated annealing [Kirkpatrick et al. 1983] is based on the analogy to the physical process of cooling and crystallizing of metals. The current state of the thermodynamic system is analogous to the current scheduling solution, the energy equation for the thermodynamic system is analogous to the objective function, and the ground state is analogous to the global optimum. In addition to the global energy, there is a global temperature, which is lowered as the iterations progress. Using this analogy, the technique randomly generates new schedules by sampling the probability distribution of the system. Since increases of energy can be accepted, the algorithm is able to escape local minima. Genetic algorithms [Goldberg 1988] are an optimization methodology based on a direct analogy to Darwinian natural selection and mutations in biological reproduction. In principle, genetic algorithms encode a parallel search through concept space, with each process attempting coarse-grain hill climbing. Instances of a concept correspond to individuals of a species. Induced changes and re-combinations of these concepts are tested against an evaluation function to see which ones will survive to the next generation.

- Fuzzy logic

Fuzzy logic is a powerful problem-solving methodology with a myriad of applications in embedded control and information processing. Fuzzy provides a remarkably simple way to draw definite conclusions from vague, ambiguous or imprecise information. In a sense, fuzzy logic resembles human decision making with its ability to work from approximate data and find precise solutions. Unlike classical logic which requires a deep understanding of a system, exact equations, and precise numeric values, Fuzzy logic incorporates an alternative way of thinking, which allows modeling complex systems using a higher level of abstraction originating from our knowledge and experience. Fuzzy Logic allows expressing this knowledge with subjective concepts such as very hot, bright red, and a long time which are mapped into exact numeric ranges. Fuzzy logic can be useful in modeling and solving job shop scheduling problems with uncertain processing times, constraints, and setup times [Grabot and Geneste1994; Tsujimura, et al.1993]. These uncertainties can be represented by fuzzy numbers that are described by using the concept of an interval of confidence. These approaches usually are integrated with other methodologies (e.g., search procedures, constraint relaxation).

2.3 Simulation as a tool for manufacturing systems scheduling

2.3.1 Simulation of manufacturing system

Simulation has been applied to long-term planning and design of manufacturing systems. These models are seldom used after the initial plans or designs are finalized [Thompson 1994]. Recent applications of simulation for real-time, operational control include emulation of real-time control systems, adaptive scheduling and planning, real-time displays of system status, performance forecasting, as well as actual implementation into a shop floor controller [Smith et al. 1994; Jones et al. 1995]. The use of simulation has appeared favorable to purely analytical methods which often fail to capture complex interactions of a particular manufacturing system.

The following are some of the specific issues that simulation is used to address in manufacturing [Law and McComas 1999]:

<u>The need and quantity of equipment and personnel</u>

- Number, type, and layout of machines for a particular objective
- Requirements for transporters, conveyors, and other support equipment (e.g., pallets and fixtures)

- Location and size of inventory buffers
- Evaluation of a change in product volume or mix
- Evaluation of the effect of a new piece of equipment on an existing manufacturing system
- Evaluation of capital investments
- Labor-requirements planning
- Number of shifts

<u>Performance evaluation</u>

- Throughput analysis
- Time-in-system analysis
- Bottleneck analysis

<u>Evaluation of operational procedures</u>

- Production scheduling
- Inventory policies
- Control strategies [e.g., for an automated guided vehicle system (AGVS)]
- Reliability analysis (e.g., effect of preventive maintenance)
- Quality-control policies

2.3.2 Simulation-based scheduling

Traditionally, scheduling problems have been solved by analytical methods, such as mathematical programming and network theory. These methods attempt to solve a problem using analytical techniques but often fail to catch the appropriate level of detail. For instance, relaxation techniques have limited practical application because they assume that processing times, down times, success rates, etc., are deterministically known. Queuing theory methods can model the steady-state operation, but are difficult to model transient situations due to the assumptions made to improve computational efficiency. The inability in developing an accurate set of constraints makes the numerical approach complex and expensive for use in large systems. These approaches often result in a static schedule. Special run-time conditions are poorly handled or not handled by many numerical approaches. Simulation is found to be very effective in the design, implementation, and operation of FMS. It has the advantages of being useful in the study of both steady state and dynamic system behavior. Moreover, it can be used as a decision support system for real-time scheduling of manufacturing systems [Miller and Pegden 2000].

In simulation-based scheduling, decision rules are incorporated into the model to make machine selection and routing decisions. The simulation constructs a schedule by simulating the flow of work through the facility and by making smart decisions based on the scheduling rules specified. There are two types of decision rules that can be applied as each job step is scheduled: an operation selection rule and a resource selection rule. If a resource becomes available and there are several operations waiting to be processed by the resource, the operation selection rule is used to select the operation that is processed next. If an operation becomes available and it can be processed on more than one resource, the resource selection rule is used to decide which resource is used to process the operation. Together these rules determine the nature and quality of the final schedule. Because of the inherent complexity and variety present in manufacturing processes, there are a large number of rules that can be applied within a simulation model to generate workable schedules. Some of these rules are focused on objectives such as maximizing throughput, maintaining high utilization on a bottleneck, minimizing changeovers, or meeting specified due dates.

2.3.3 Real-time simulation-based scheduling of FMS

2.3.3.1 Real-time simulation concept

A real-time application is an application where the correctness of the application depends on the timelines and predictability of the application as well as the results of computations. Real-time applications provide an action or an answer to an external event in a timely and predictable manner. In general, FMSs are more sensitive to system disturbance than conventional manufacturing systems because of tighter synchronization, system integration, and interdependencies among automated components. Hence, they require an immediate response to changes in system states, such as arrival of parts, machine status, queues at machines, tool breakages, rushed jobs, and many other system disturbances. Real-time activities in manufacturing environment primarily refer to daily operations that require efficient, timely, and adaptive responses to short-term planning, scheduling, and execution problems. Pertinent areas of interest include job releases, loading sequences, deadlocks, and response to resource disruptions such as machine or tool failure [Drake et al. 1995]. Practically, the speed at which a decision is needed to be considered real-time is dependent upon system parameters such as the magnitude of part processing times and the flexibility of the system [Harmonosky and Robohn 1991]. If processing times are on the order of an hour, a response in 5 minutes may pass for real-time; if processing times are on the order of a few minutes, real-time responses are probably needed in less that 1 minute. Another differentiation exists in the literature between continuous real-time scheduling, where every decision regarding which task to schedule next is made as needed as time moves forward in the physical system, and exception real-time scheduling, where a real-time scheduling decision is only made when a machine breakdown, part shortage, or the like occurs. Whether a real-time scheduling methodology can be considered as a candidate for application depends upon a particular system's definition of real-time.

2.3.3.2 Real-time simulation based FMS scheduling

The FMS is modeled as a set of interconnected queuing systems, 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. 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 work-piece has been fully processed, it exits the FMS immediately. 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 available workstation at some decision points depending on dispatching rules.

In an ideal system, operations are computer controlled, setups between consecutive operations are automated, and most operations are processed by numerical control (NC) machine tools, so processing times are nearly deterministic. This implies that the schedule result is predictable if there are no system disturbances. In some situation, therefore, fixed off-line scheduling may be enough. The dynamic and uncertain nature of system states, however, may make off-line scheduling impractical for FMS. Unlike a typical job shop, the NC machines have a number of features such as automatic tool interchange capability and automatic material handling, which can permit very general flow patterns, but also induces new problems for management and control. More scheduling decisions must be considered including machine setup, part routing and operation sequencing. Further complexity is introduced because the need of consideration of other constrained resources in the system, such as material handling devices, storage buffers and tool magazines. In general, FMSs are more sensitive to system disturbance than conventional manufacturing systems because of tighter synchronization, system integration, and interdependencies among automated components. Hence, they require an immediate response to changes in system states, such as arrival of parts, machine status, queues at machines, tool breakages, rushed jobs, and many other system disturbances.

Real-time control of an FMS is not an easy job. Flexible routings, processing, and part mix, as well as the dynamic nature of a shop floor, place tremendous demands upon the control system. A detailed understanding of operational information is required for efficient production. In any attempt to use simulation as a real-time scheduling tool, the issues of how to initialize the simulation to current shop status when a decision is needed and how to determine an appropriate look-ahead window length and parameters must be considered. The look-ahead window length for simulations evaluating several alternatives at a real-time decision point has a direct impact upon potential real-time application of simulation. Longer run lengths should provide statistically better output, but that makes decision response time longer. Harmonosky and Robohn [1995] investigate how physical system parameters affect the amount of time it takes to perform the simulations given a specific level of precision for statistical accuracy of the simulation output.
In most of the research using simulation for real-time scheduling, the simulation is assumed to be interfaced with the physical system in some manner. Figure 2.1 presents a typical viewpoint for the interface for exception real-time scheduling. Data regarding system status updates the simulation initial conditions when a decision is needed, either on an exception basis or continually. If alternatives exist, then each would be simulated and the best option could be selected [Harmonosky 1990]. Two approaches are generally used to carry out the simulation-based scheduling task: continuous scheduling approach and exception scheduling approach. Continuous real-time scheduling refers to the case where every decision regarding which task to schedule next is made as needed as time moves forward in the physical system. Exception real-time scheduling refers to the case where a real-time scheduling is only made when a disruption occurs, such as a machine breakdown or part shortage.

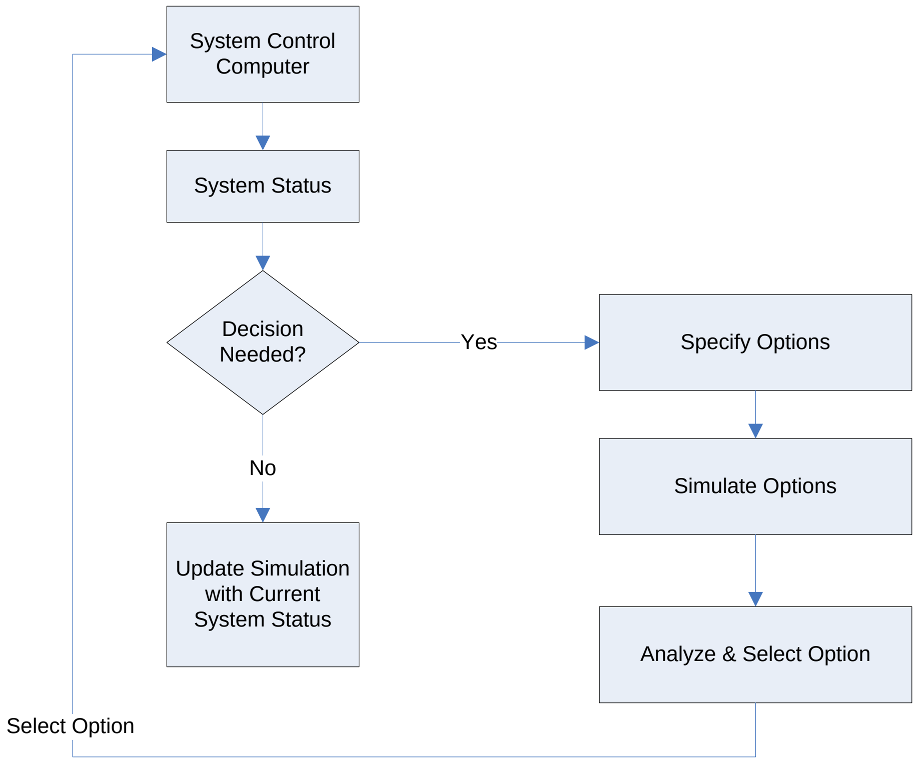


Figure 2.1: Framework of exception real-time scheduling

CHAPTER 3 ADVANCED SIMULATION MODELING TECHNOLOGIES FOR FMS SCHEDULING

3.1 Parallel discrete event simulation

3.1.1 Discrete event simulation

A discrete event simulation (DES) is one in which system state changes only at a set of discrete points in time called event times. Between two successive event times, systems state does not change. A discrete event simulation is carried out by increasing simulated time in some manner and updating system state when necessary [Banks and Carson 1986]. DES is one way of building up models to observe the time based (or dynamic) behavior of a system. During the experimental phase the models are executed (run over time) in order to generate results. The results can then be used to provide insight into a system and a basis to make decisions on. Discrete event simulations typically have three basic common denominators. First, they contain a set of state variables denoting the current state of the simulation. The state variables contain information such as the number and availability of system resources. Secondly, a typical discrete simulation contains an event queue. The event queue is a list of pending events which have been created by an event generator but not yet executed by the scheduler. These events require system resources to execute. The availability of resources is described by the state variables. Events often contain arrival timestamp and event duration indicating service time. The arrival timestamp indicates when the event impacts the system's state variables. Event arrival times and service times are frequently generated based on statistical models. For example, events may arrive according to a Poisson distribution. Finally, the third common denominator of discrete event simulations is the global simulation clock which keeps track of the simulation's progress. The simulation must maintain proper causal states, meaning that each event must be executed in the environment created by the execution of the prior events.

3.1.2 Introducing parallelism --- PDES

Recent 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 [1979]. 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 [Fujimoto 2000].

3.1.2.1 Model decomposition

Most research efforts attempting to improve the simulation technology were carried out in the studies of parallel discrete-event simulation (PDES). The basic idea is to decompose a simulation model into sub-models and distribute the simulation computations on different processors concurrently. 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 or central control among processes in PDES. 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.

- Multiple Replication Simulation

An execution of a simulation model can be viewed as computing a sample path for a random process X = {x(t), t≥0}. The simulation result is an estimation of the steady-state mean of state variable x as $E(x)=\frac{1}{T}\int_0^T x(t)d(t)$, where T is the simulation time period of one sample. 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 [Heidelberger 1986].

- Time Segmentation Simulation

Another idea of parallel simulation can be referred as time segmentation simulation [Wang and Abrams 1992]. Basically, the simulation time period t is divided into several time intervals, and each time segment is simulated concurrently on one processor. Because of its time-domain partitioning, time-parallel simulation can potentially yield massive parallelism. 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.

- Space Parallel Simulation

The majority of parallel simulation methods are space parallel simulation, which basically decompose a simulation model into a number of components from the space domain of the model [Page and Nance 1994]. For example, a simulation model for an FMS can be divided into several sub-models, in which each sub-model 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.

3.1.2.2 Synchronization problem

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 [Dado 1993]. The causality principle states the fact that the future cannot be affected by the past. 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. To address this issue, the concept of local causality constraint was proposed by Fujimoto [2000] which states:

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

If each LP adheres to the local causality constraint, then the parallel execution will yield the correct results. 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 is not violated. To meet such a requirement, two timing mechanisms, the conservative and the optimistic parallel simulation approach, were developed [Reynolds 1988]. The conservative approach strictly avoids causality errors at any time point during the simulation by implementing a global clock. 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 [Fujimoto 1993]. One difficult problem in this approach is the avoidance of deadlock. Various execution mechanisms have been proposed to avoid and/or detect deadlock, including null messages, circulating marker [Misra 1986], and time warp [Jefferson et al. 1987]. Another approach is named optimistic. 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 [Jefferson 1985] is the first optimistic parallel simulation approach.

3.2 Traditional hardware platforms for PDES

PDES models are usually built on hardware platforms which contain a potentially large number of processors interconnected through a communication network. In most cases the processor is a general purpose CPU. The communication network may be as specific as a customized switch for a particular multi-processor system, or as general as the Internet. Commonly used platforms are summarized as follows.

3.2.1 Shared-memory multiprocessors

The distinguishing property of the programming model for shared-memory multiprocessors is one may define variables that are accessible by different processors. Shared variables and message passing are the two dominant forms of inter-processor communications used in PDES. One type of shared-memory machine, the symmetric multiprocessor (SMP), has become increasingly popular. These systems consist of off-the-shelf microprocessors connected to memory through a high speed switch, such as bus. Frequently accessed instructions and data are stored in a high speed cache memory that is attached to each processor. Typically the multiprocessor hardware automatically moves data and instructions between the cache and main memories, so the programmers need not be concerned with its operation. Consistency protocols are required to ensure that multiple copies of any shared variable residing in different caches remain up-to-date if one copy is modified. A second class of shared-memory multiprocessor is called non-uniform memory access (NUMA) machine. These machines are typically constructed by providing memory with each processor but allow each processor to directly read or write the memory attached to anther processor. Unlike symmetric multiprocessors, the programmer's interface to these machines distinguishes between local and remote memory, and provides faster access to local memory. Both conservative and optimistic PDES have been realized on shared-memory multiprocessors [Reed et al. 1987; Konas and Yew 1991; Teo at al. 2002].

3.2.2 Distributed memory multi-computers

Multi-computers do not support shared variables. Rather, all communications between processors must occur via message passing. Large multi-computers may contain hundreds of processors. Unlike the cache used in shared-memory multiprocessors, the cache in each multi-computer node only holds instructions and data for the local processor, so no cache coherence protocol is needed. The memory in each node can only be accessed by the CPU in that node. The communication controller is responsible for sending and receiving messages between nodes, and typically transfers messages directly between that node's memory and the inter-connection network. Such transfers that usually do not require the intervention of the CPU are referred to as direct memory access (DMA) operations. One of the earliest distributed simulation model based on multi-computer platform was proposed by Kumar [1986].

3.2.3 SIMD machines

SIMD stands for single-instruction-stream, multiple-data-stream. The central characteristic of these machines is that all processors must execute the same instruction at any instant in the program's execution. Typically these machines execute in "lock-step", synchronous to a global clock. This means all processors must complete execution of the current instruction before any is allowed to proceed to the next instruction. SIMD machines typically contain more, albeit simpler, processors (processing elements) than either multiprocessors or multi-computers. Because they are simpler, custom-designed components are usually used rather than off-the-shelf parts. [Ayani and Berkman 1993; Greenberg et al. 1996].

3.2.4 Distributed computers

Two characteristics that distinguish distributed computers from parallel machines are heterogeneity and the network used to interconnect the machines. Unlike parallel computers, distributed systems are often composed of stand-alone computer workstations from different manufacturers. Heterogeneity is important because many distributed simulators are constructed by interconnecting existing sequential simulators operating on specific workstations. Heterogeneity eliminates the need to port existing simulators to new platforms, and it enables participation of users with different computing equipment in distributed simulation exercise. Distributed computers use general interconnects based on widely accepted telecommunication standards such as asynchronous transfer mode (ATM). This kind of platform has been widely studied in large scaled simulation like PADS, HAL and Web-based parallel simulation [Page 1999].

3.3 Hardware acceleration of discrete event simulation

3.3.1 ASIC assisted DES

Previously some efforts have been made to utilize some specially designed hardware function unit to accelerate non-deterministic discrete event simulation speed. As an example, Fujimoto [1993] designed a special hardware component called the rollback chip (RBC) to address the state saving function. In optimistic algorithm, time warp method relies on a rollback mechanism to undo the effect of clock synchronization errors. However, the state saving overheads in rollback algorithms are serious, and will cripple Time Warp programs containing large amounts of state. Rather than copying data into memory, the rollback chip manipulates addresses generated by the CPU to avoid overwriting data that may later be required after a future rollback operation. The rollback chip is one component of a special purpose parallel discrete event simulation engine which uses a message-based multi-computer architecture. It implements state saving and rollback function for a single processor in the simulation engine, and results showed that it can significantly lower the cost of state saving. Reynolds [1992] proposed a novel framework for providing rapid dissemination of critical synchronization information in all parallel discrete event simulation implementations. This framework can produce a significant reduction in the finishing times of parallel simulations. The strength of this framework lies in its use of special purpose, high-speed hardware, specifically a *parallel* *reduction network* (*PRN*) which computes and disseminates results of *global reduction operations*. A global reduction operation is a binary, associative operation performed on data across all processors. Although special-purpose ASIC circuits can be designed to replace some functional parts of simulation, this method is not widely applied due to issues regarding to cost and flexibility.

3.3.2 Field programmable gate array (FPGA) assisted DES

Due to the inherent inflexibility of ASIC, researchers have turned to reconfigurable devices for performance enhancement. A detailed survey in this area can be found in Bumble [2001]. The following sections briefly describe some featured achievements recently.

3.3.2.1 Deterministic logic machines

Special purpose machines have been designed and implemented specifically for the development and performance enhancement of logic systems. In logic design, simulations are used to verify new projects and to run fault analysis of these designs. The logic simulation is usually employed to study deterministic system behavior.

The Reconfigurable Machine (RM) combines FPGAs and RAMs to support a wide range of applications [Tomita et al. 1993]. The RM incorporates FPGAs which are capable of in-circuit reconfiguration allowing the RM to reload several types of configuration data during power-on (Figure 3.1). It is an event-driven, fine-grained, conservative logic simulator. The RM architecture employs FPGAs which allow in-circuit reconfiguration and relatively fast switching speeds. One of the FPGAs serves as the interface module. The other four FPGAs serve as processing modules. Each of the four FPGAs can access both shared and distributed memory.

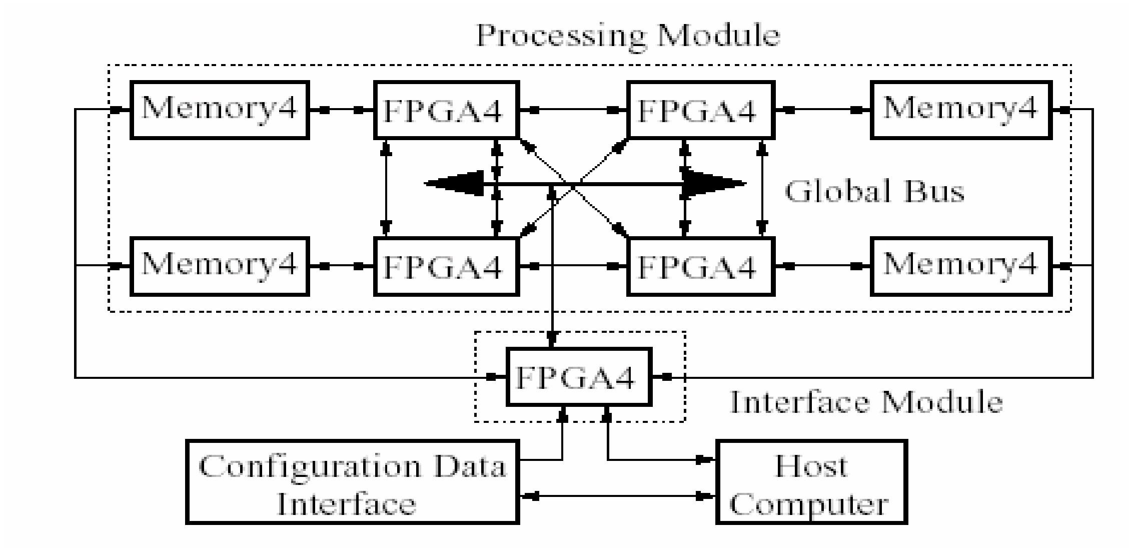


Figure 3.1: Reconfigurable Machine by Tomita et al. [1993]

A more recent logic simulation design which utilizes FPGAs was developed in work by Bauer [1998]. This logic simulator uses reconfigurable logic to accelerate the discrete event simulation of logic circuits. The focus of this work is accelerating discrete event simulation, but the target is again deterministic. The foundation for this reconfigurable computing system is an FPGA-based emulator, which provides large blocks of reconfigurable logic (Figure 3.2). Each emulation module runs a small operating system to manage the behavioral simulation and logic net-list emulation. A separate control processor which is not illustrated performs higher level operating system functions including network access and disk management. The emulation modules consist of a PowerPC 403GCX processor, local RAM, and a local FPGA array with its associated programmable interconnect. Emulation modules connect to each other via programmable interconnects.

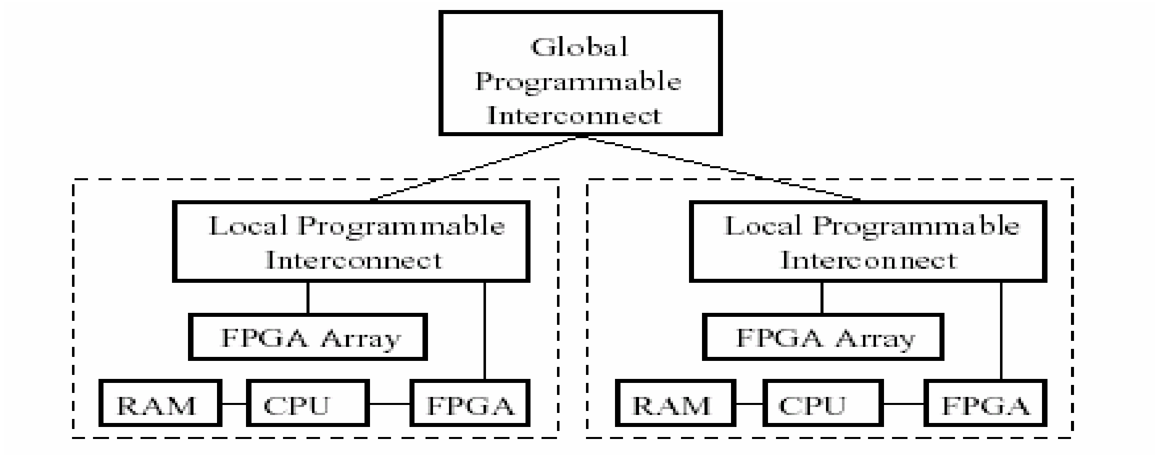

Global Programmable Interconnect
Local Programmable Interconnect
FPGA Array
RAM
CPU
FPGA
Local Programmable Interconnect
FPGA Array
RAM
CPU
FPGA

Figure 3.2: Reconfigurable Logic Simulator by Bauer et al. [1998]

3.3.2.2 General purpose machine

- Splash

Splash was designed to serve as a systolic processing system using a Sun workstation as its host [Arnold et al. 1992]. The general purpose machine is normally an SIMD machine and the original design was motivated by a systolic algorithm for DNA pattern matching (Figure 3.3).

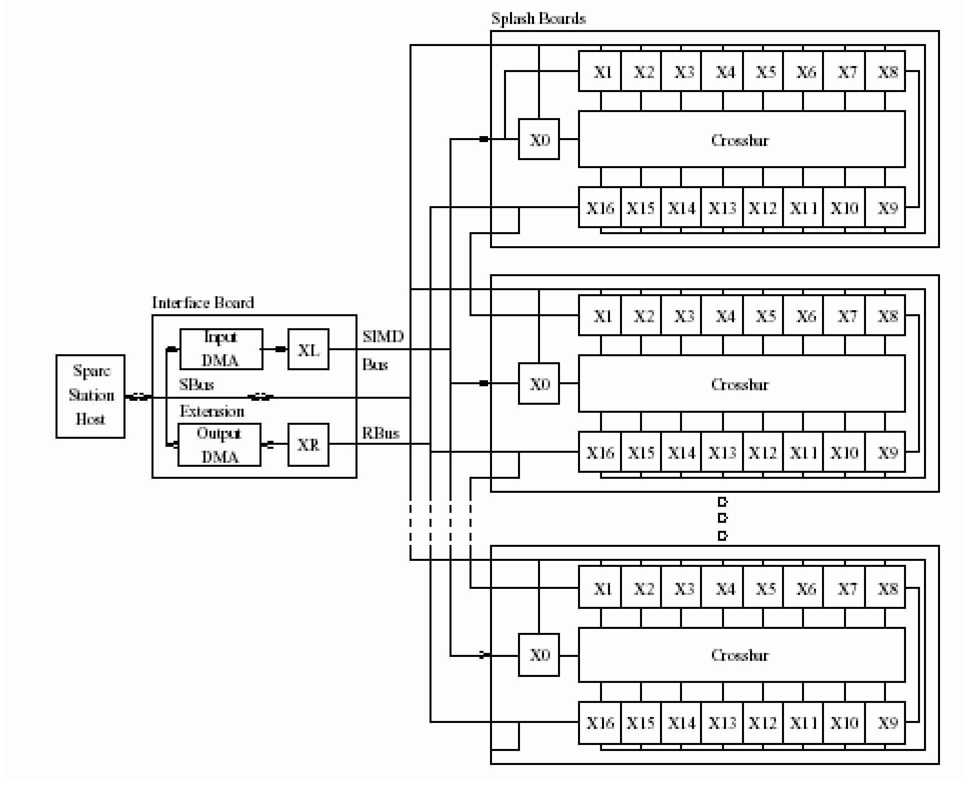


Figure 3.3: The Splash 2 Architecture by Arnold et al. [1992]

Splash was designed to handle various programming models including a SIMD model, a one-dimensional pipelined systolic model, and several higher-dimensional systolic models. The SIMD applications utilize the X0 element and the crossbar switch on each board to broadcast the instructions and data to all processing elements simultaneously. The instruction stream is sent from the host to the X0 chip on each board via the SIMD bus. X0 broadcasts the instruction to all 16 of the board's processing elements, which are each programmed with one or more identical SIMD computing engines. These engines synchronously receive and execute instructions and perform nearest neighbor communications through the linear data path. Global element synchronization is accomplished with the AND/OR reduction network. One-dimensional systolic arrays are formed by using the processing board's 36-bit linear data path to form a continuous pipeline from the host, through the array, and back to the host. The crossbar switch allows an individual processing element to be bypassed or multi-dimensional systolic arrays to be implemented.

- The ArMen

Beaumont et al. [1994] proposed a new architecture for discrete synchronous event-driven simulation using FPGAs. The MIMD ArMen implementation allows the parallel execution of events with the same timestamp in virtual time. The machine is an event-driven, fine-grained, conservative simulator (Figure 3.4). All processors wait until all the event computations for a given simulation cycle are complete. Then the simulation can proceed to the next phase. The next simulation cycle is the global minimum of all the minimum time-stamped events on each node. The protocol respects causality constraints since all processors are always executing events with the same timestamp. The two main global control operations are the synchronization barrier which every processor must reach before the simulation can proceed to the next simulation cycle, and the calculation of the global minimum of all the Local Virtual Times (LVTs) in order to determine the next time to be simulated. These two operations have been implemented in the FPGAs of the ArMen machine.

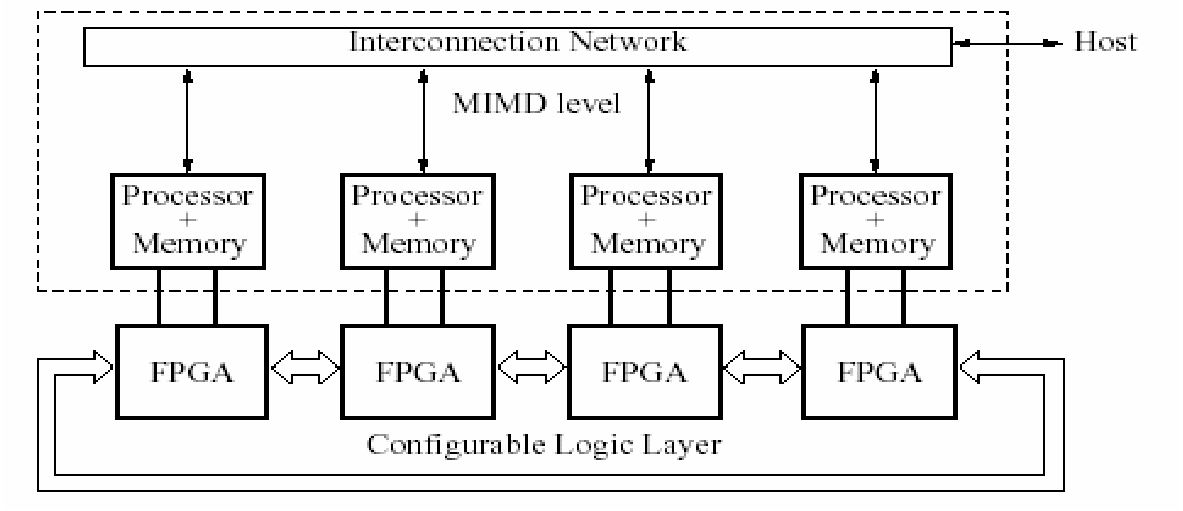


Figure 3.4: The ArMen Architecture by Beaumont et al. [1994]

- Marc Bumble

Bumble and Coraor [1998] presented a conservative simulator which is significantly accelerated by its FPGA components. The simulation speedup is accomplished via two independent enhancements. The first accelerates random event generation. The second facilitates faster handling of service events generated by scheduled arrival events (Figure 3.5). The first enhancement accomplishes event generation speedup by translating some simulation loop software into parallel, systolic, and reconfigurable logic. Reconfigurable logic permits various statistical distribution models to be compiled from software into hardware implementations. The Event Generator computes event arrival times, service times, and resource requirements with some partial parallelism. Event Generation, illustrated in Figure 3.6, is subdivided into the creation of arrival and service times. Event generation is accomplished by a two-dimensional reconfigurable systolic array which allows the two time offsets to be created in parallel. The systolic array pumps data from one processing block to the next in regular time intervals, until the data circulates to the Event Queue.

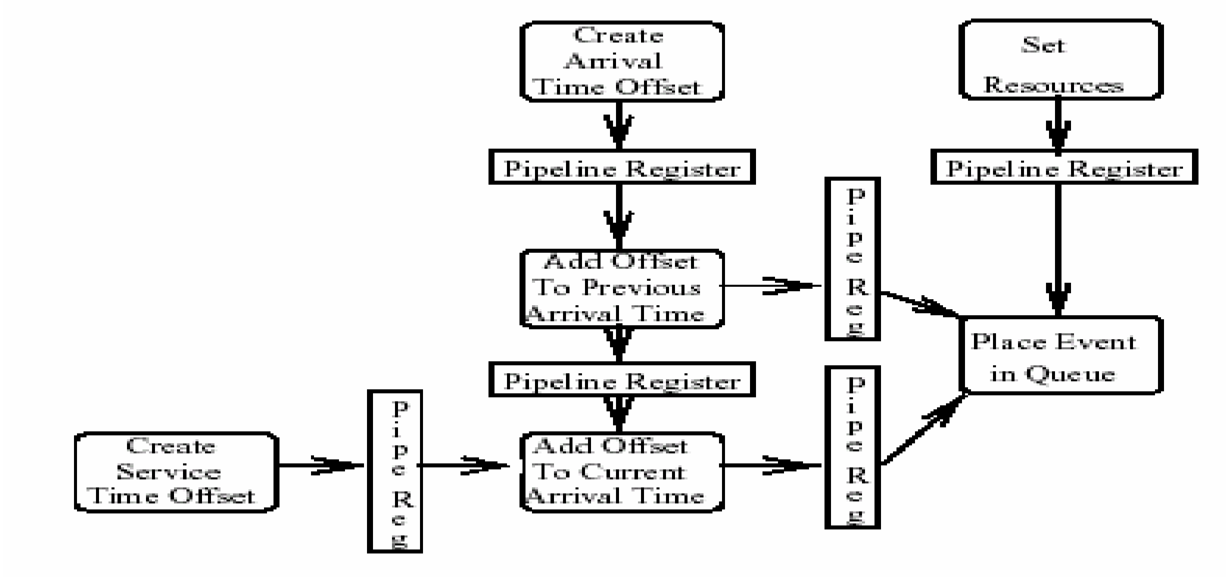

Create Arrival Time Offset
Set Resources
Pipeline Register
Pipeline Register
Pipe Reg
Add Offset To Previous Arrival Time
Place Event in Queue
Pipeline Register
Pipe Reg
Create Service Time Offset
Pipe Reg
Add Offset To Current Arrival Time

Figure 3.5: Event Generator Flow Diagram by Bumble and Coraor [1998]

The second simulation enhancement employs a split event queue. The queue stores the events in two separate queues, the *arrival* and *service* queues. The split queues are implemented as hardware, with the service queue able to select its minimum timestamp. The local processing element design uses two queues for each server. The arrival queue holds the sorted list of arrival events, which arrives in-order from the Event Generator, and the queue can be implemented as a

FIFO queue. Service events, which are created from processing successful arrival events, are stored in the service queue.

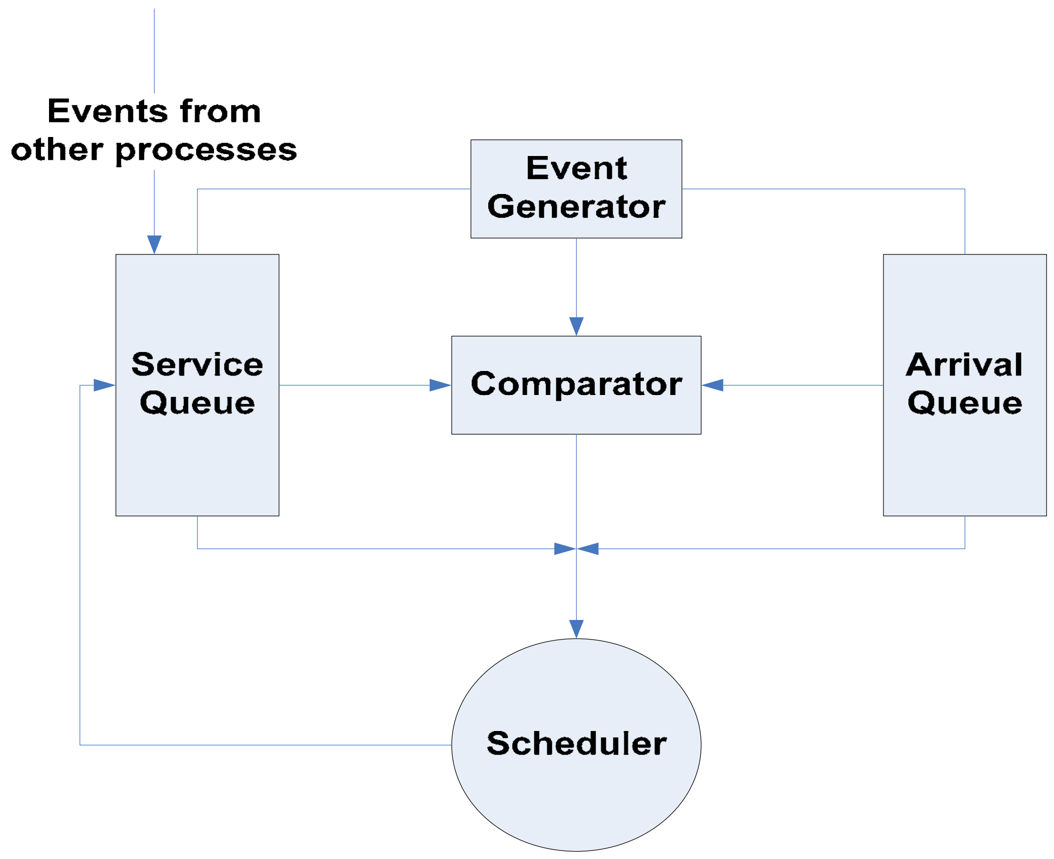


Figure 3.6: Local Processing Element adapted from Bumble and Coraor [1998]

After events are created by the event generator, they are stored in the arrival queue in order. The arrival queue can be easily implemented as a FIFO queue. Successfully executed arrival events create service events. A special algorithm is designed to maintain a sorted queue and can select the *nth* element to insert a new element. A special parallel bus architecture employing a technology such as Emitter Coupled Logic (ECL) is proposed. The primary function of the parallel bus is to locate the minimum network time stamp and synchronize the network. A secondary function of the bus allows the processing elements to communicate with the centralized controllers.

3.4 DSP-based simulator for manufacturing systems

As a former member in FMS lab at Virginia Tech and the predecessor of this research, Xu [2001] proposed a hardware-based parallel simulation method using DSP-multiprocessor system to carry out parallel simulation for FMS. 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 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.

3.4.1 Hardware-based simulator Design

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 supporting 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 key methodology for building the hardware-based simulator is mapping. 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. Mapping at the individual equipment level is based on either one-to-one reflecting or multiple-to-one relations. The framework of the parallel I/O port based architecture is shown in Figure 3.7.

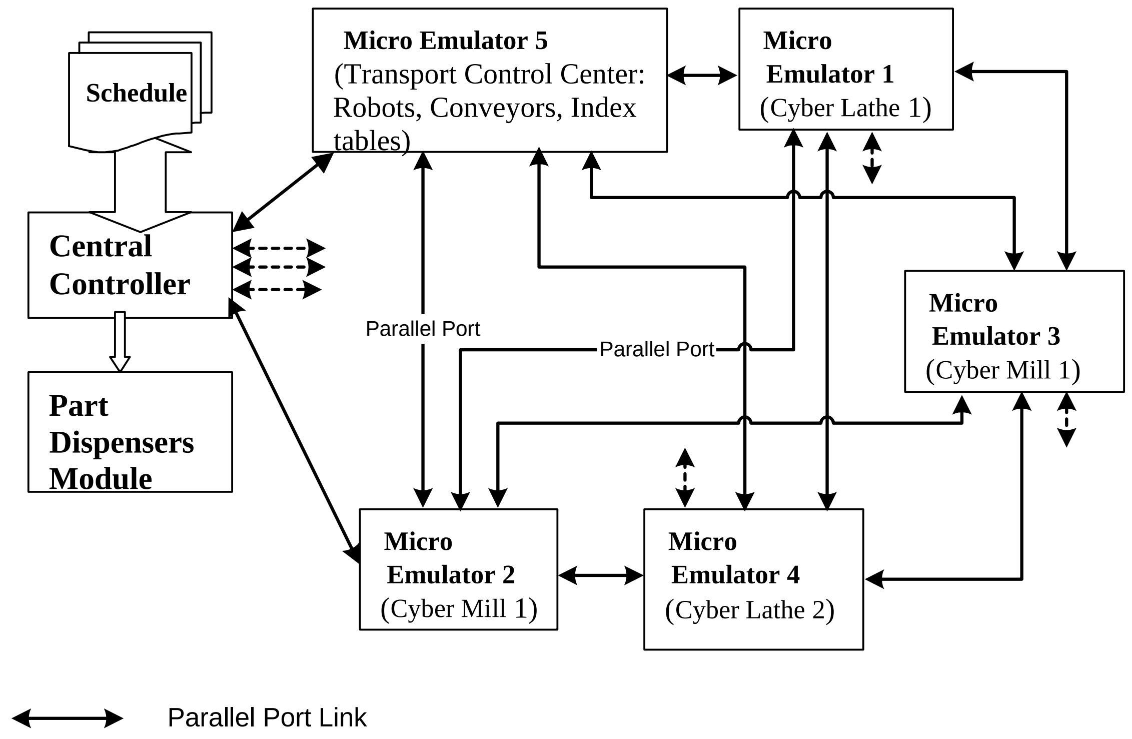


Figure 3.7: Parallel I/O port based architecture by Xu [1991]

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. Since there are limited parallel ports on a DSP chip and it is unrealistic to connect all the microprocessors by this way, an alternative design scheme has been proposed. The idea 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. To utilize state-of-the-art hardware technology advancement, emulators are built on DSP processors with powerful data processing capability.

To address the synchronization issue, a time-scaling mechanism was proposed. 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 eclipsing time period in the real world are now represented by the amount of clock cycles last on the microprocessor to represent the activity. 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.

Based on the hardware-based parallel simulation methodology, one prototype simulator has been developed (Figure 3.8). The prototype simulator was specifically designed to simulate the laboratory desktop FMS located in the FMS lab at ISE department, Virginia Tech. 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, and it leaves a large space to modify the board according to specific custom design. On each DSK board there is a TMS320C31 DSP chip, running at 50M Hz.

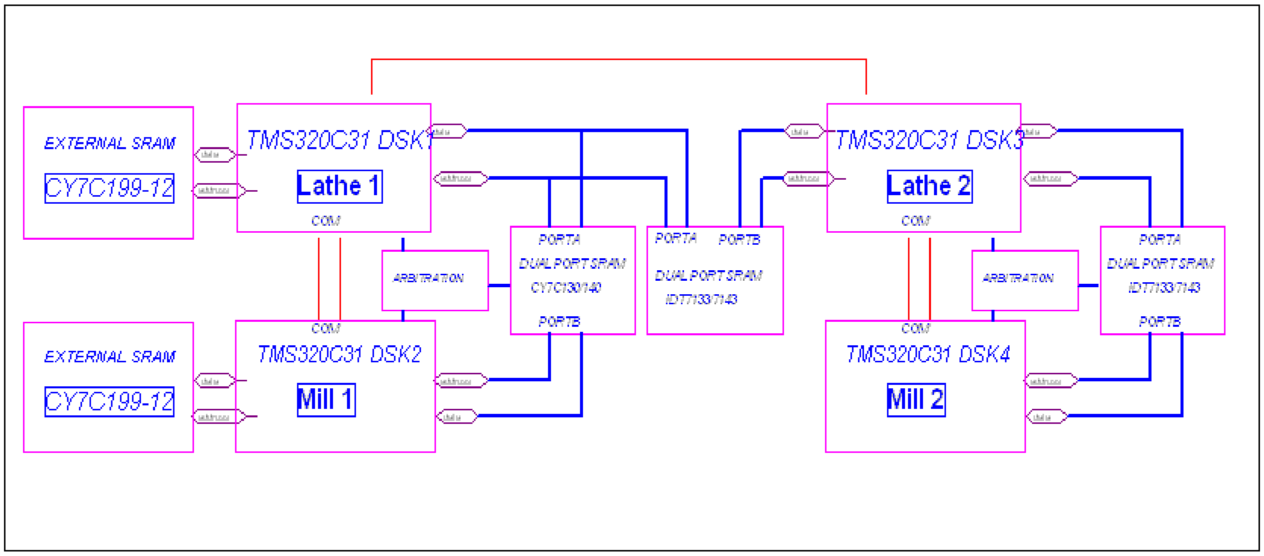


Figure 3.8: Parallel IO-based architecture implementation by Xu [1991]

3.4.2 Experiments on the simulator prototype

- Experiment Case for a 2-Workstation Production Line

In this experiment, a 2-workstation production line 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). 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.

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. The results show that 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.

- Experiment Case for 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 sub-modeling methods with the support of direct digital links were programmed. One sub-modeling method makes robots as the core objects and the other one makes workstations as the core objects. For the first sub-modeling 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 sub-modeling method focusing on workstations, 4 DSPs are used, and each is mapped to one workstation.

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.

3.4.3 Conclusion from previous research

The DSP-based simulator 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 a great potential to achieve a dramatic speedup 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. But the method 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 why the hardware-based parallel simulation is a worthwhile endeavor.

3.5 Reconfigurable hardware-based simulation modeling for manufacturing systems

3.5.1 Traditional software based discrete event simulation modeling methodology

3.5.1.1 Simulation modeling of FMS

Modern high-tech manufacturing systems can be extremely complex. The complexity of these systems is due to factors such as: multiple part types made in the same facility, numerous manufacturing steps, batch processing, system complexity which leads to high costs of maintenance, multiple levels of subassemblies, etc. This complexity combined with the high cost of setting up and maintaining such a system necessitates the use of formal models of the system, rather than just relying on experience or simple rules of thumb for performance evaluation and decision making.

Models are intended to support decisions about the system and a single model usually is not capable of supporting all decisions. Rather, different decisions require different models because various aspects of the design and operation of the system are important for the questions being asked of the model. While spreadsheet and queuing models are useful for answering basic questions about manufacturing systems, discrete event simulation models are often needed to answer detailed questions about how a complex manufacturing system will perform. Simulation models have the ability of incorporating additional details about the manufacturing system and therefore often give more accurate estimates of manufacturing system behavior than the simpler models mentioned above, but usually at the cost of more computation. In general, simulation is a practical methodology for understanding the high-level dynamics of a complex manufacturing system.

3.5.1.2 Modeling methodology

Many different methodologies and tools are available to support manufacturing projects, including spreadsheets, analytic models, data-driven simulation models, simulation languages, and general-purpose programming languages with simulation libraries [Chance, et al. 1996].

Spreadsheet models are appealing for projects where a large number of scenarios must be evaluated quickly. They are fast, easy to understand and use, and easy to modify. Managers are generally accustomed to reviewing results presented in spreadsheet-based formats. However, there are limits to the types of questions that can be addressed with spreadsheet models.
Spreadsheet models are static, generally grouping time into large buckets. They are also usually deterministic. Though spreadsheets typically include the ability to sample from distributions, this capability is not usually exploited in manufacturing applications. Because of these limitations, spreadsheet models are often not capable of estimating cycle time or WIP.

Analytic models include capacity analysis, queuing, linear programming, and other math programming models. Like spreadsheet models, analytic models are useful for evaluating multiple scenarios, because they are typically very fast. Analytic models can also include detail beyond that of spreadsheets. The results from analytical models are relatively easy to interpret, because they consist of a single number for each performance measure. However, the simplifying assumptions necessary to get closed form results are not always appropriate, especially in queuing models. Analytic models are best suited to strategic and tactical questions, where the emphasis for dynamic behavior is on relative performance.

Discrete event simulation can capture virtually any level of manufacturing detail, and is potentially very accurate. It captures both dynamic and stochastic behavior. Simulation is sometimes used in the early stages of a project to help understand how the system works. It has intuitive appeal for managers, especially when animation is used, because they can “see” what is going on in the model. However, there are several disadvantages to using simulation models. They typically take much longer to run than analytic models, and the results from a simulation model can be difficult to interpret. Statistical analysis of the output is necessary, because each simulation run represents a single possible sample path. For these reasons, simulation can be an expensive option.

Manufacturing simulation models can be developed using both general-purpose and manufacturing-focused tools. However, depending on the complexity of the system being modeled, manufacturing focused tools can significantly simplify and quicken the modeling process. There are many different manufacturing-oriented simulation packages on the market and each has its strengths and weaknesses. Some of the more popular manufacturing-oriented packages include Arena, AutoMod, ProModel, and Witness.

3.5.1.3 Basic simulation modeling methodology inside DES

There are several basic elements inside a traditional discrete event simulation:

Entities: Most simulations involves "players" called entities that move around, change status, affect and are affected by other entities and the state of the system, and affect the output performance measures. Entities are the dynamic objects in the simulation. They usually are created, move around for a while, and then are disposed of as they leave. It is possible though, to have entities that never leave but just keep circulating in the system. However, all entities have to be created, either by the modeler or automatically by the software.

Attributes: To individualize entities, attributes are attached to them. An attribute is a common characteristic of all entities, but with a specific value that can differ from one entity to another.

Variables: A variable is a piece of information that reflects some characteristics of the target system, regardless of how many or what kinds of entities might be around. There are many different variables in a model, but each one is unique.

Resources: A resource can represent a group of several individual servers, each of which is called a unit of resource. Entities often compete with each other for service from resources that represent things like personnel, equipment, or space in a storage area of limited size.

Queues: When an entity cannot move on, perhaps because it needs to seize a unit of a resource that is tied up by another entity, it needs a place to wait, which is the purpose of a queue. Queues have names and can also have capacities to represent.

Other elements include statistical accumulators, event, simulation clock, etc [Kelton, el al. 2002]. Most widely used commercial software packages feature similar elements.

The implementation of a DES model can take a variety of forms. One of the implementation details for a discrete event simulation is the mechanism through which simulation time is advanced. Two basic forms exist:

Fixed-time increment (or time-stepped): Here the values assumed by the simulation clock are evenly spaced along an interval on the real numbers.

Variable-time increment (or next event, event-stepped): Here the values assumed by the simulation clock are spaced arbitrarily apart.

Regardless of the time flow mechanism utilized, the event remains the fundamental concept that binds time and state; one view (next-event) considers the passage of time to be predicated on events, the other (fixed-time) considers events to be predicated on the passage of time.

Discrete event simulation models and their implementations may reflect any of a variety of organizing principles. For example, a simulation may be oriented around events - as provided by languages like SIMSCRIPT. Or it may be oriented around processes (lifetime of objects within the model). The organizing principle of the model is known as its conceptual framework (or world view). Two most recognized conceptual frameworks are event scheduling and process interaction. A correctness-preserving DES implementation preserves the notional execution described by the state chain above. Event scheduling languages directly implement this execution scheme. In a process oriented language, such as MODSIM or SIMULA, the attribute value changes at one time are often happening among several objects. The event is harder to distinguish in a process oriented simulation language, since it is distributed among the object lifetimes. Still, the execution of a process oriented simulation preserves the notional DES execution. An event scheduling and process oriented implementation of the same model will produce exactly the same results.

3.5.2 Innovative simulation modeling technology for FMS

This research proposes an entirely new simulation modeling technology, which neither relies on computer nor software package. Instead, it uses electronic signals and circuits to represent the behavior of manufacturing system. The system is based on synchronous circuits, in which all activities inside the model are driven by a global digital clock. System states could only change at the rising edge of global clock. With the introduction of reconfigurable computing technology, the model can be easily built and altered according to specifications. To our knowledge, no previous research has been done in this area. The realization of some common system features are described below.

- Entities

Entities in the simulation model are represented by electronic signals transmitting through various functional units via physical link. For example, a part is modeled as an 8-bit binary signal which contains part type and operation index information. The part signal can be further extended to include more attributes, such as due time information.

- Entity inter-arrival times

The entity inter-arrival times are modeled by a hardware random signal generator. The generator can produce a series of part signals with time distance following some pre-defined pattern. For example, suppose the target part inter-arrival follows exponential distribution. The random signal generator will be able to generate a series of uniform random numbers, convert them to exponential distributed random variates, and then use them to control the timing of part signal generation.

- Queues

Queues are modeled by synchronous FIFO circuits. The circuit can be realized either by Linear Feedback Shift Register (LFSR) or memory. It can write a number of part signals and read them out according to writing order. It also can provide status signals such as “empty”, “full” or “data count”.

- Resources

Resources, such as machine centers or material handling system are modeled by specifically designed circuitry.

Simulation clock will be realized by a physical clock, which provides clock signal to all functional units. All events in the simulation are synchronized by this global clock signal. There is no event list inside the simulation model, because any event will be carried out immediately when it appears. Therefore, no waiting is needed.

3.5.3 Applications in real-time rescheduling of FMS

3.5.3.1 Reactive scheduling in FMS

Reactive scheduling, or rescheduling, is the process of updating an existing production schedule in response to disruptions or other changes, including the arrival of new jobs, machine failures, and machine repairs. The performance of a production system depends not only on the original planning and scheduling, but also on good and proper rescheduling with the uncertainties present in the production environment. These uncertainties often result in orders following a route through the shop floor different from the one originally developed. In such cases, previously generated schedules become invalid and have to be regenerated or revised. Rescheduling usually is invoked at the time a disturbance occurs and takes into account the current state of production on the shop floor. An acceptable revised schedule is the one that overcomes the new constraints, for example, a schedule that allows re-routing parts from a machine that has just broken down. In practice, rescheduling may be done periodically to plan activities for the next time period based on the state of the system. It may also be done occasionally in response to significant disruptions. Because time estimates are inaccurate and unexpected events occur, precisely following a schedule becomes more difficult as time passes. In some cases, the system may follow the sequence that the schedule specifies even though the planned start and end times are no longer feasible. Eventually, however, a new schedule will be needed. Vieira et al. [2003] presents a framework for understanding rescheduling not only as a collection of techniques for generating and updating production schedules but also as a control strategy that has an impact on manufacturing system performance in a variety of environments. The framework includes rescheduling environments, rescheduling strategies, rescheduling policies, and rescheduling methods. The rescheduling environment identifies the set of jobs that need to be scheduled. A rescheduling strategy describes whether or not production schedules are generated. A rescheduling policy specifies when rescheduling should occur. Rescheduling methods describe how schedules are generated and updated.

Rescheduling can be carried out either manually or through application software. The manual method involves editing the existing schedules which are normally in the form of a Gantt chart. This method is tedious and time consuming. Researchers have often used simulation for rescheduling purposes. Yamamoto and Nof [1985] used a “regeneration” method in developing their scheduling systems. This method involves rescheduling the entire set of operations including those unaffected by the change in conditions, demands, or constraints. This is time consuming, and often results in response times unacceptable to the user. They compared three scheduling procedures to deal with machine breakdowns. Matsuura et al. [1993] used simulation to investigate rescheduling, and approached the problem by selecting between sequencing and dispatching in case of uncertainties. A schedule is first determined by using a branch and bound technique. This approach is switched to dispatching, which uses either the FCFS or SPT rule, when there is a change in production conditions. Li et al. [1993] proposed a rescheduling algorithm based on the construction of a scheduling binary tree and a net change concept adopted from MRP systems. Dutta [1990] proposed a knowledge-based system to help automate the control activity at the scheduling level in FMS environments. The author assumed a batch size of one which is rarely the case in actual production systems. Jain and Elmaraghy [1997] used Genetic algorithms to obtain an initial schedule and considered four different types of uncertainties that normally cause discrepancies between the actual output and the planned output in rescheduling process, including unforeseen machine breakdowns, increased order priority, rush orders arrival and order cancellations.

3.5.3.2 A simulation based rescheduling mechanism

Two broad categories of reactive scheduling approaches can be identified as predictive-reactive scheduling and dynamic scheduling [Vieira, et al 2002]. The predictive-reactive approach seeks to establish an order for all the open jobs and reacts to process disturbances by reordering the jobs, while the dynamic approach provides a solution by the use of dispatching rules for selection from the queue of jobs. Predictive-reactive scheduling is a two-stage approach, the first stage generates a schedule and the second updates the schedule in response to disruptions. Schedule generation acts as a predictive mechanism for production activities, and is important to serve as the basis for planning support activities such as material procurement, etc. Updating the schedule can involve either partial repair of the disrupted schedule or complete rescheduling. The update can be either periodic, event-driven or hybrid. A periodic policy regenerates schedule periodically. An event-driven policy performs rescheduling upon the occurrence of events. A hybrid policy, as a combination of both periodic and event driven policies, reschedules the system periodically or when major events take place.

Dynamic scheduling does not create production schedules. Instead, decentralized production control methods dispatch jobs when necessary and use information available at the moment of dispatching. Such schemes use dispatching rules or other heuristics to prioritize jobs waiting for processing at a resource. Dynamic scheduling schemes sometimes are referred as on-line scheduling or reactive scheduling. Dispatching rules and pull mechanisms are used to control production without a production schedule. When a machine becomes available it chooses from among the jobs in its queue by using a dispatching rule that sorts the jobs by some criteria. Common dispatching rules employ processing times and due dates in simple rules and complex combinations. Some dispatching rules are extensions of policies that work well on simple machine scheduling problems (e.g., Shortest Processing Time and Earliest Due Date). The computational effort of dispatching rules is low when simple rules are used. However, some dispatching rules require a large amount of information, and the job priorities must be recalculated at every dispatching decision. Dynamic scheduling is closely related to real-time control, since decisions are made based on the current state of the manufacturing system. Therefore, the reconfigurable hardware-based simulation proposed in this research is well suited for such applications.

Kim and Kim [1994] developed a scheduling mechanism based on the scheduling/rescheduling approach. In the mechanism, job dispatching rules are selected according to results from discrete event simulation. There are two major components, a simulation mechanism and a real-time control system. The simulation mechanism evaluates various dispatching rules and selects the best one for a given criterion. The real-time control system periodically monitors the shop floor and checks the system performance value. The best dispatching rule is used until the difference between the actual performance value and the value estimated by simulation exceeds a given limit; then a new simulation is performed with remaining operations in the simulation mechanism, and a new rule is selected. The simulation mechanism includes a model constructed by information from decisions of the planning problem. When the rule selector calls the simulation mechanism, a series of discrete event simulations is performed with each rule in the dispatching rule set under the same conditions. With the outputs from the simulation, the rule selector selects the best one for a given performance measure. Then the selected rule goes to the scheduling controller and becomes an input to the control system. Jobs are dispatched according to this rule at the shop floor. At this moment, the scheduling mechanism saves the performance value that is estimated from the selected dispatching rule. This estimated value is another output from the simulation mechanism. The real-time control system sends all scheduling controller information to the shop floor and dispatches jobs to machines accordingly. Also, the real-time control system periodically monitors and checks the actual performance value of the shop floor. If the difference between actual and estimated performance values exceeds a predetermined limit at a point of monitoring, a new series of discrete event simulations is performed with the remaining operations under the current state to select a new dispatching rule. If the difference is within the performance limit, the remaining process is continued with the current rule.

Chong, et al [2003] further extended researches in this area, and proposed a simulation based scheduling mechanism, which dynamically adapts to the changing manufacturing condition. In their approach, discrete-event simulation is used both off-line and on-line. Simulation is used off-line to build reference indices based on the performance of different scheduling approaches under varying shop floor conditions. Simulation is then used on-line to evaluate the schedules generated by the better scheduling approaches for the actual manufacturing conditions to identify the best to use.

Based on above research, a new simulation based scheduling mechanism is proposed (Figure 3.9)

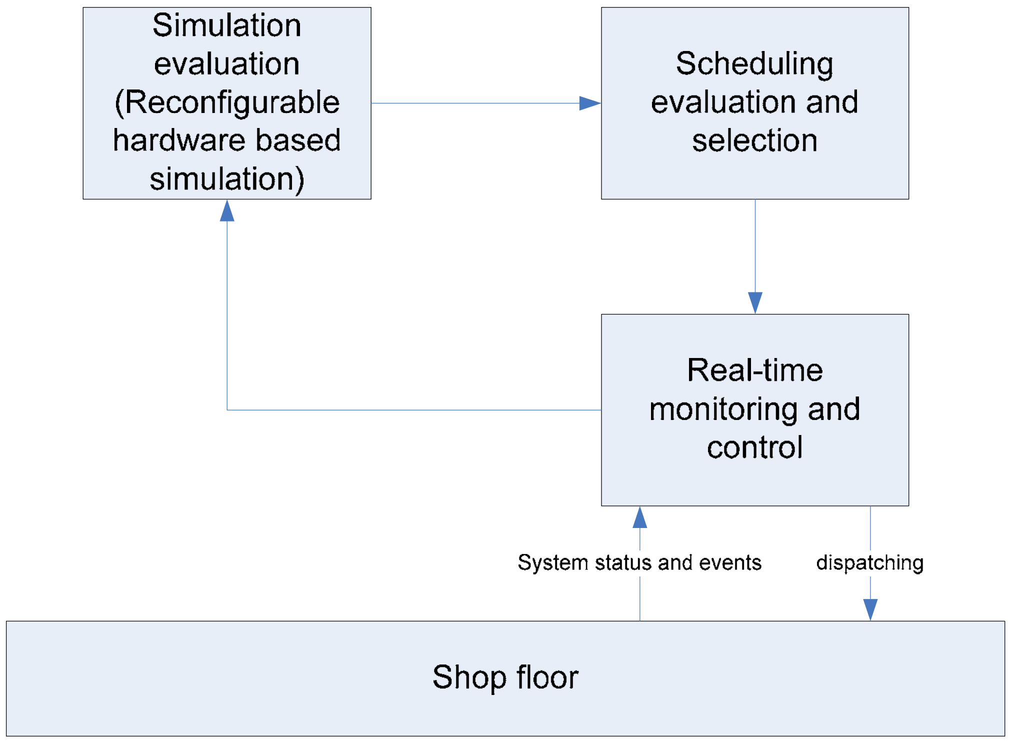

Simulation evaluation (Reconfigurable hardware based simulation)
Scheduling evaluation and selection
Real-time monitoring and control
System status and events
dispatching
Shop floor

Figure 3.9: Reconfigurable hardware simulation based rescheduling

Real-time monitoring and control module receives events from the shop floor, and periodically monitors performance of the shop floor. It also sends all scheduling controller information to the shop floor and dispatches jobs to machine accordingly. Two sub-components are included: disturbance analysis and performance analysis. The former checks all incoming events, invokes rescheduling on major disturbances. The latter monitors the difference between actual and estimated performance values. Once the difference exceeds a predetermined limit at a point of monitoring, a new rescheduling is performed with the remaining operations under the current shop floor conditions. The simulation evaluation module includes a reconfigurable hardware based simulation model constructed according to the physical shop floor status from the factory database. When the scheduling controller passes control to this module, a series of simulation runs are initialized with the different dispatching rules under the same stochastic conditions. The length of simulation is set to be equal to the length of scheduling window. The results of the simulation will be passed back to the controller. The outputs from the simulation are used by scheduling controller to selects the best schedule. The schedule evaluation and selection module selects the best dispatching rule to be used on the shop floor. This selection process is performed at the beginning of a rolling scheduling horizon, when the system performance is not as expected, or when there is a major disturbance. The selection is also performed when the system state is back to normal (e.g., at the time of machine operating after repaired).

CHAPTER 4 HARDWARE SIMULATOR DESIGN METHODOLOGY

4.1 Motivation for a full hardware simulator

In previously discussed DSP-based simulator, some interesting ideas had been proposed, including system mapping, time scaling, digital signal represented parts, etc. However, due to the limitation of DSP microprocessor, those ideas were not able to be fully realized in the design. Basically, DSP-based simulator is still a microprocessor based system bearing some inherent weaknesses of general microprocessor that may not suit for real-time processing. In this section, the shortcomings of DSP-based simulator are discussed and an initiative of designing a full hardware simulator is described.

4.1.1 Limitations of DSP-based simulator

- Limited speedup

One of the design goals is to dramatically increase simulation speed. DSPs provide some techniques to expedite data processing, such as superscalar and VLIW [Lyons 1996]. Those functions are initially designed for vocal signal processing in communication area, which involves massive simple but repetitive data processing. In some aspects, DSP can help the computation in simulation run, such as random number generating, routing algorithm computing, etc. However, DSP-based simulator is still a microprocessor based simulator which must rely on software to perform most of its tasks. For example, during a simulation run, if Part A at Machine 1 is to be sent to Machine 2, then DSP1 should fetch Part A variable from register, and write it to shared memory. DSP2 then reads the shared memory, and if there is no contention then the part transportation could be completed. This process may be realized in a number of instructions, and each instruction takes a number of clock cycles. This is certainly not very efficient. Therefore, DSP-based simulator may only be able to achieve rather limited speedup. Another disadvantage of DSP lies in the limited parallelism it can achieve. From the architectural standpoint, DSP processes data in a sequential manner. Though via some techniques like flow line processing, DSP can realize some kind of temporal parallelism, yet it can not employ massive parallelism whenever the situation calls for it.

- Computation barrier

In simulation execution the clock is scaled down in proportion to real world time, so it can be called a faster-than-real-time simulation. It is like a fast forward of a movie, in which everything goes by quickly but still maintains the temporal order, and the local causality constraint is naturally held. The prerequisite of this kind of fast play of simulation is that every processing time in the simulation can be scaled down proportionally. Unfortunately, some processes in real life are difficult to scale down, for example, the process of routing algorithm computation. In real life, such computation may take only seconds in a cell controller. To scale down this computing time by a large factor is certainly unrealistic, unless we have a processor thousands of times more powerful. Some unconventional strategies, such as stopping global clock until computation is finished, were proposed in previous research [Xu 2001]. However, this makes the simulation control very complicated and unreliable.

- Lack of scalability

The DSP-based simulator is designed on a parallel IO based architecture. This provides the machine centers a fast way of transporting parts, for each pair of machines own their dedicate channel in which the parts are moved around with a fixed transmission delay. This method is efficient when the system remains at small scale. However, when the system becomes larger, the number of channels will grow exponentially, making the system overly complex. In other words, the system does not possess good scalability. In real world, manufacturers may upgrade their manufacturing system by adding some equipment in order to meet new production requirements. A simulator should be capable of quickly incorporating the changes made to the real system accordingly. That implies a non-scalable simulator model might be inappropriate.

- Insufficient resources

Since DSP is designed for general purpose applications, it only has a limited number of built-in functional units, such as memory, timers and counters. Therefore, the majority of simulation functions still have to be realized by software programs. For example, in time-scaling simulation, each event should use a timer to trigger its activities. If multiple events are processed simultaneously, then a large number of timers will be needed. Usually a DSP only provides two hardware timers. Therefore, programmed delays have to be used to emulate the behavior of timer. This will significantly drags down the simulation speed.

4.1.2 Full hardware simulator based on reconfigurable computing architecture

Traditionally there are two primary methods in conventional computing for the execution of algorithms. The first one is to use hardware technology, either an Application Specific Integrated Circuit (ASIC) or a group of individual components forming a board-level solution, to perform the operations in hardware. ASICs are designed specifically to perform a given computation, and thus they are very fast and efficient when executing the exact computation for which they were designed. However, the circuit cannot be altered after fabrication. This forces a redesign and re-fabrication of the chip if any part of its circuit requires modification. It is an expensive process, especially when one considers the difficulties in replacing ASICs in a large number of deployed systems. Board-level circuits are also somewhat inflexible, frequently requiring a board redesign and replacement in the event of changes to the application. The second method is to use software-programmed microprocessors --- a far more flexible solution. Processors execute a set of instructions to perform a computation. By changing the software instructions, the functionality of the system is altered without changing the hardware. However, the downside of this flexibility is that the performance can suffer, either in clock speed or in work rate, and is far below that of an ASIC. The processor must read each instruction from memory, decode its meaning, and only then execute it. This results in a high execution overhead for each individual operation. Additionally, the set of instructions that may be used by a program is determined at the fabrication time of the processor. Any other operations that are to be implemented must be built out of existing instructions. Reconfigurable computing is intended to fill the gap between hardware and software, achieving potentially much higher performance than software, while maintaining a higher level of flexibility than hardware. Reconfigurable devices, including field-programmable gate arrays (FPGAs), contain an array of computational elements whose functionality is determined through multiple programmable configuration bits. These elements, sometimes known as logic blocks, are connected using a set of routing resources that are also programmable. In this way, custom digital circuits can be mapped to the reconfigurable hardware by computing the logic functions of the circuit within the logic blocks, and using the configurable routing to connect the blocks together to form the necessary circuit [Compton and

Hauck 2002]. From above analysis, it becomes obvious that DSP-based system can only achieve limited simulation speedup. An alternative thought is to bypass the microprocessor and software programs and realize all functionalities via reconfigurable devices. In such a simulator, the microprocessor is no longer a required part. Instead, all major simulation functions and algorithms are realized via hardware connection, and entities will be represented by electronic signals.

4.2 Introduction to reconfigurable computing technology and devices

4.2.1 Reconfigurable computing concept

The concept has been in existence for quite some time [Estrin et al. 1963]. Reconfigurable computing is defined as computing via post-fabrication, spatially programmed connection of processing elements. It can simply be thought of as an ability to repeatedly configure a machine to perform different and varying functions [Bhatia 1997]. As an illustration of spatial programmed connection, Figure 4.1 demonstrates how the parallel execution advantage can be used. The software routine needs 12 clock cycles to calculate the result G; however, in hardware it takes only 2 clock cycles to compute the same result [DeHon 1999].

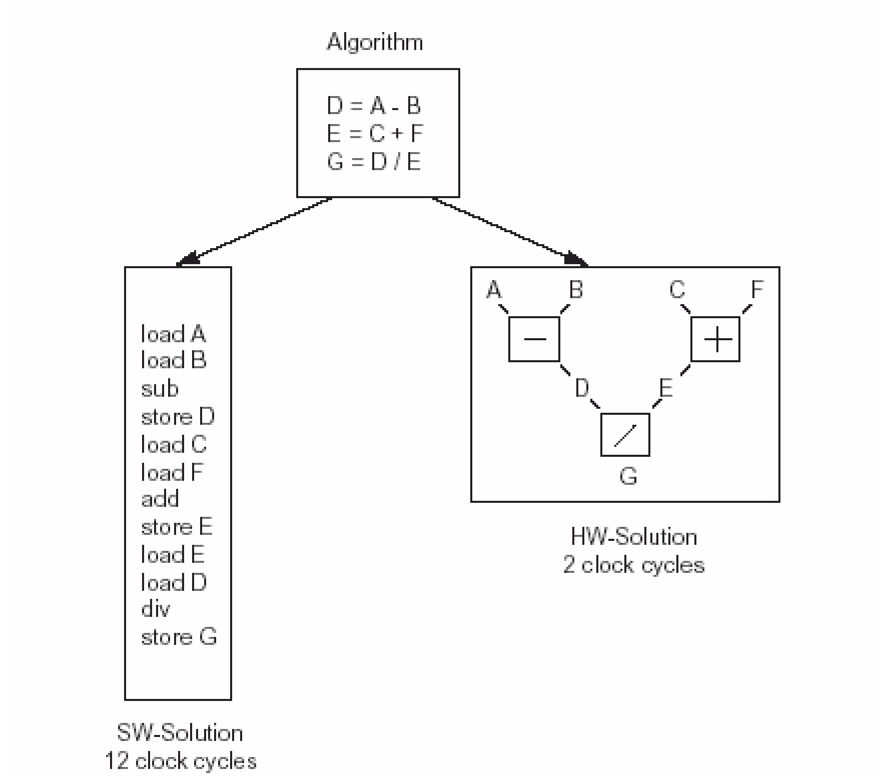


Figure 4.1: Space parallelism

Usually configurable computing devices refer to those devices which can be programmed by end users according to their specific purpose. The devices may be reprogrammed for multiple times off-line. An example of such device is EPROM. Re-configurable means the configuration can be performed at run-time. Therefore, reconfigurable computing combines features of data processor and ASIC approaches. The realization of reconfigurable systems was enabled through the introduction of the field-programmable gate array (FPGA) in the mid eighties. FPGAs share with ASICs the capability to implement application-specific circuits, with the key difference that FPGA circuits are programmed by means of a configuration data stream that specifies the logical functionality and connectivity. Although originally proposed in the late 1960s, reconfigurable computing is a relatively new field of study. The decades-long delay had mostly to do with a lack of acceptable reconfigurable hardware. Reprogrammable logic chips like field programmable gate arrays (FPGAs) have been around for many years, but these chips have only recently reached gate densities making them suitable for high-end applications. With an anticipated doubling of gate densities every 18 months, the situation will only become more favorable from this point forward.

4.2.2 Field Programmable Gate Arrays

FPGA is an integrated circuit that contains many (64 to over 10,000) identical logic cells that can be viewed as standard components. Each logic cell can be independently wired to realize some limited functions. The individual cells are interconnected by a matrix of wires and programmable switches. A user's design is implemented by specifying the simple logic function for each cell and selectively closing the switches in the interconnect matrix. Simpler functional blocks can be combined to realize complex designs [Rose et al. 1993]. FPGA devices can be re-configured to change logic function while resident in the system. This capability gives the system designer a new degree of freedom not available with any other type of logic. Hardware can be changed as easily as software. Design updates or modifications are easy, and can be made to products already in the field. An FPGA can even be reconfigured dynamically to perform different functions at different times. Reconfigurable logic can be used to implement system self-diagnostics, create systems capable of being re-configured for different environments or operations, or implement multi-purpose hardware for a given application. As an added benefit, using reconfigurable FPGA devices simplifies hardware design and debugging and shortens product time-to-market [Hauck, 1998].

Xilinx is the most prominent FPGA manufacturer in the market. Its Virtex series FPGA products feature a flexible, regular architecture that comprises an array of configurable logic blocks (CLBs) surrounded by programmable input/output blocks (IOBs), all interconnected by a rich hierarchy of fast, versatile routing resources. The abundance of routing resources permits the Virtex family to accommodate even the largest and most complex designs. Virtex FPGAs are SRAM-based, and are customized by loading configuration data into internal memory cells. In some modes, the FPGA reads its own configuration data from an external PROM (master serial mode). Otherwise, the configuration data is written into the FPGA (Select-MAP™, slave serial, and JTAG modes). Xilinx also provides a powerful developing environment called ISE Foundation, which offers a variety of tools to meet the developer’s needs. For example, it includes many intellectual properties (IPs) which realize some specific functions. These intellectual properties have been developed and tested by the manufacturer, thus can save the developers a lot of efforts. Currently, Virtex FPGA is selected for this simulator design [Xilinx 2005 (3)].

4.3 An innovative simulation paradigm

It is widely believed that synchronization techniques impose too much overhead to be useful in real-time distributed simulation. In many cases, the load imposed by time management algorithms is not predictable. Total overhead and predictability are two major impediments to the adoption of time-management techniques in the real-time distributed simulation community. To address these issues, a new simulation paradigm is proposed.

4.3.1 System representation

In such a scheme, entities inside the system are no longer represented by variables stored in memory. Instead, they are represented by electronic signals being transmitted through gate level circuitry. It is well known that the travel speed of the electronic signal is among the fastest in the universe. Thus by mapping the processes in a manufacturing system to the electronic signal activities inside a circuit, immense simulation speedup can be achieved. Decision makings in manufacturing system will be modeled by basic gate activities like AND, OR and NOT. Data processing functions can be vastly distributed to millions of gate level logic devices running concurrently. Traditional operational activities in a manufacturing system can be described as follows: First, hardware random number generator provides an inter-arrival time following some specific distribution. Then the arrival time is stored in a timer and counted. As time is up, a pulse (representing part) is sent to an FIFO queue which represents the buffer of a machine center. If machine status signal indicates that the machine is currently empty, the signal moves out of the FIFO queue and enters machine processing unit. Meanwhile another timer begins to work, representing the part processing time. Again, when processing time is up, the part signal will be sent out of the machine and transmitted to other machines for processing. Hundreds of such part signals travel in the circuit simultaneously, and every event occurs following its timer. All timers are synchronized by a global clock. This scenario is just like people living in the real world, where everyone doing individual business while also interacting with each other. However, inside the simulator, events occur and are processed at a lightening fast speed. Since massive parallelism is employed this way, one can image that thousands of times of simulation speedup can be expected.

4.3.2 Time flow mechanism

There are several different notions of time that are important when discussing a simulation:

- *Physical time* refers to time in the physical system
- *Simulation time* is an abstraction used by the simulation to model physical time.
- *Wall-clock time* refers to time during the execution of the simulation program.

Simulation executions where advances in simulation time are paced by wall-clock time are often referred to as real-time simulation. A variation is scaled real-time simulation where simulation time advances faster or slower than wall-clock time by some constant factor. The following function can be used to convert wall-clock time to simulation time:

$$T_s = T_{start} + Scale * (T_w - T_{wstart})$$

Simulation time, T_s, equals to start time T_{start} plus scaled wall-clock time duration.

In a discrete simulation, the simulation model views the physical system as only changing state at discrete points in simulation time. The two most common types of discrete simulations are called time-stepped and event-driven simulation. These two categories of simulation are distinguished by time flow mechanism. In a time-stepped simulation, simulation time is subdivided as a sequence of equal-sized time steps, and the simulation advances from one time step to the next. A new state for the simulation is calculated at each step, although not every state variable needs to be modified. Actions in the simulation occurring in the same time step are considered to be simultaneous, and are often assumed not to have an affect on each other. Rather than compute a new value for state variables each time step, it may be more efficient to only update the variable when some event occurs. This is the idea leading to discrete event simulation. In a discrete event simulation, simulation time advances from the time stamp of one event to the next. From a computational standpoint, the simulation can be viewed as a sequence of computations, one for each event, transforming the system across simulated time in a manner representing the behavior of the actual system. An event-driven simulation can emulate a time-stepped simulation by defining events that happen at each time step, for example, events can be defined with time stamp 1, 2, 3…, assuming a time step size of one. A variation of the event-driven paradigm can also be used for real-time or scaled real-time simulations. One approach is to prevent simulation time from advancing to the time stamp of the next event until wall-clock time has advanced to the time of this event.

If one combines the key features of the above two variations, a very special time flow mechanism can be inferred. In such mechanism, simulation time advances in time steps, such as 1, 2, 3…, only when scaled wall-clock time reaches next event time. In order to assure every logic process is synchronized, all process must keep in pace with a global clock. Such kind of synchronization has been referred to as Single Clock Multiple Systems (SCMS) simulation [Vakili 1992]. Based on the combination of this special time advancement mechanism and the unique system representation described in last section, a new simulation paradigm is proposed.

4.3.3 Synchronous real-time simulation --- SRS

Although the time advancement mechanism of this newly proposed simulation paradigm can be regarded as a very special variation of event-driven, the simulation itself is hard to be classified as PDES, for it lacks some of the fundamental characteristics of PDES. For example, there is no event list in the proposed simulation paradigm, so efforts for event list managing are saved; there are no procedures for state computing and event scheduling either, because these are fulfilled by specific hardware connection; and the tremendous synchronization overhead is avoided by introducing a global clock. Therefore, in this research the new simulation paradigm will be referred to as *synchronous real-time simulation* or SRS.

One of the unique features of SRS is using electronic signals and circuits to mimic behaviors of entity and workstation in a manufacturing system. These mimicking functions are realized through sequential logic circuitry implemented in an FPGA chip. Each machine center can be modeled by a circuit unit, and multiple units can be grouped to form a manufacturing cell. The circuitry is referred to as hardware simulator. It can run stand-alone without connecting to microprocessors in order to achieve maximum speed. Simulation time is in proportional to real-world wall-clock time, therefore all local processes are naturally synchronized by a global clock. The simulation is inherently deadlock free because no local process has to wait for messages from other processes before it proceeds. Clock skew is the major issue to be considered, so it is preferable that the simulation is parallel but not distributed, and all local processes should be located adjacently so that the clock skew is not significant in affecting the simulation results. To achieve maximum execution speed, simulation time is scaled down by a large factor. The time scaling factor will be determined by the trade-off of required simulation speed and accuracy. Any events happening within the same time interval will be regarded as simultaneous events.

Specific tie-breaking rules will be applied to these events. The comparison between parallel discrete event simulation and synchronous real-time simulation is summarized below.

	PDES	SRS
Speed	low	fast
Parallelism	space and time parallel	massively space parallel
Modeling approach	asynchronous	synchronous
Time advancement	event-driven	time-stepped
Deadlock free	no	yes
Synchronization Mechanism	conservative or optimistic	global clock
Flexibility	high	medium
Algorithm realization	software procedure	hardware connection
Hardware utilization	high	low
Hardware overhead	low	high
Simulation accuracy	high	adjustable

Table 4.1: PDES vs. SRS

4.4 System design methodology

In this new simulator design, the system mapping concept is inherent from previous DSP-based simulator, which basically decomposes a simulation model into a number of components from the space domain of the model. In this case, a simulation model for an FMS is divided into several sub-models, in which each sub-model 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 (machine center emulator), which is responsible for managing those state changes for the corresponding component over the simulation time period.

4.4.1 System architecture

4.4.1.1 System mapping

The target desktop FMS layout is shown in Figure 4.2. The system consists of the 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)

The simulation model is built to mimic the desktop FMS located in the FMS lab at Virginia Tech. There are four machine centers in the FMS, two lathes and two milling machines. Their functionalities are mapped into four individual emulator units. At the first stage, some peripheral equipment like robot and inspection stations are omitted in order to grasp the main characteristics of the manufacturing system. The focus is on the behaviors of machine centers and part flow. Other device models will be added in a step-by-step manner. The new simulator is no longer a multi- microprocessor based system. As a result, each machine center is mapped to a hardware emulator unit which can mimic the production activities via electronic signals and circuitry. There are no programs describing the functions of each machine center, instead, the functions are reflected in the hardware connection. System inputs/outputs are in the form of electronics signals and binary digits.

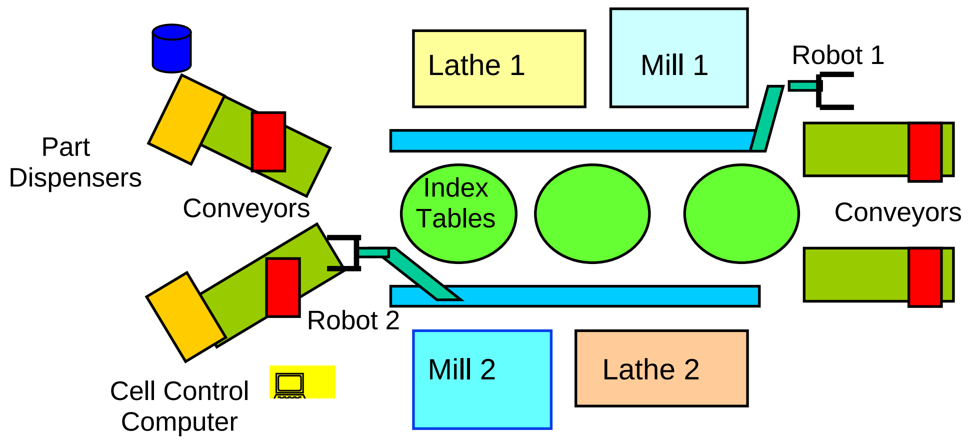


Figure 4.2: Desktop Flexible Manufacturing System

4.4.1.2 Star topology network architecture

The previous DSP-based simulator has been considered not scalable due to its parallel IO based system architecture. A star topology is designed with each node connected directly to a central network hub or concentrator. Data on a star network passes through the hub or concentrator before continuing to its destination. The hub or concentrator manages and controls all functions of the network. It also acts as a repeater for the data flow. Since robots play a key role in material flow inside the system, it seems to be more reasonable to model the manufacturing cells as star topology network.

Advantages of a Star Topology

- Easy to install and wire.
- No disruptions to the network when connecting or removing devices.
- Easy to detect faults and to remove parts.

Disadvantages of a Star Topology

- Requires more cable length than a linear topology.
- If the hub or concentrator fails, nodes attached are disabled.
- More expensive than linear bus topologies because of the cost of the concentrators.

As manufacturing system becomes larger, the star topology network will evolve into a tree topology network. A tree topology combines characteristics of linear bus and star topologies. It consists of groups of star-configured workstations connected to a linear bus backbone cable. This feature ensures the scalability of the model, which is vital for manufacturing system modeling.

4.4.1.3 The role of microprocessor

Although the simulator can run stand-alone, some secondary functions may be suitable to be realized in microprocessor, such as system initialization, run-time control and reporting. Programmable logic tends to be inefficient at implementing certain types of operations, such as loop and branch control. In order to most efficiently run an application in a reconfigurable computing system, the areas of the program that cannot be easily mapped to the reconfigurable logic are executed on a host microprocessor. Meanwhile, the areas with a high density of computation that can benefit from implementation in hardware are mapped to the reconfigurable logic. In hardware simulator design, the need of a general-purpose microprocessor lies in several observations: First, microprocessor provides a user friendly interface through which people can easily perform simulator reconfiguration, such as selecting routing algorithms, adding machine centers or changing system parameters. Second, microprocessor offers an efficient way of simulation initialization and activity control. It can handle supporting tasks such as generating random number seeds, setting simulation time and replications, as well as modeling non-critical signals like machine down, etc. Third, microprocessor can provide plenty of powerful mathematical functions which are needed for statistics computing and reporting. As for the coupling mechanism, due to the large speed discrepancy of the two, a loosely coupled scheme is selected in order to achieve maximum parallelism. The whole system architecture is illustrated in Figure 4.3.

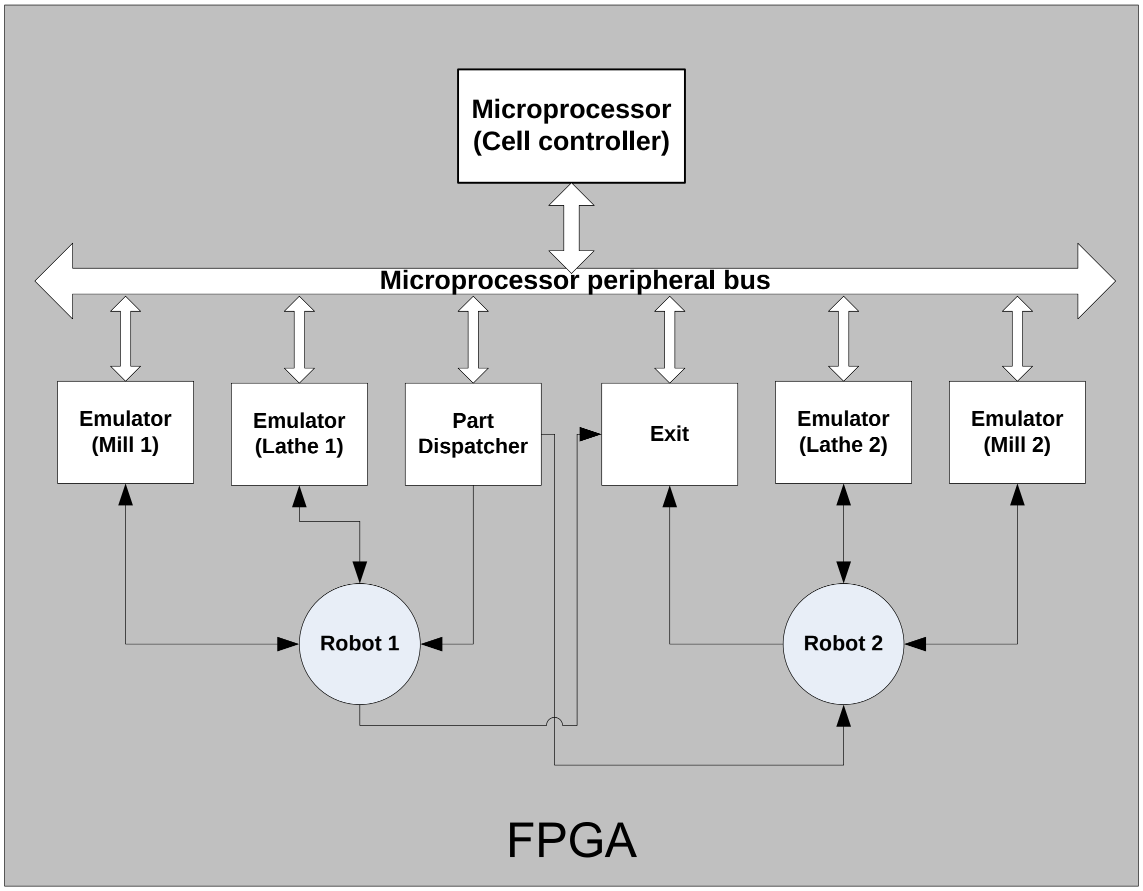


Figure 4.3: Hardware simulator system architecture

4.4.2 Emulator unit design

In a traditional discrete event simulation model, functions could be divided into four parts: simulation clock, the event generator, the event queue and the event processor. However, in this special variant of DES, there will be no event queue, since any event will be processed immediately after it is generated. Moreover, there is no simulation clock management because the simulation time of each local process is driven by global clock. Therefore, the only two components to be realized in hardware design are event generator and event processor, in the form of random signal generator and event processing unit, respectively. The random signal generator produces a series of electronic signals, representing entities inside the system. Events are generated as these traveling signals arrive at their processing places (machine, queue, or router, etc). The core of an event processing unit (mapping of a machine center) is a timer, and it counts the clock cycles needed for event processing then releases the resources and sends the signal away.

4.4.2.1 Random signal generator

Random signal generator consists of a multi-bit linear feedback register (LFSR) based random generator, a reverse function converter, a buffer and a timer (Figure 4.4).

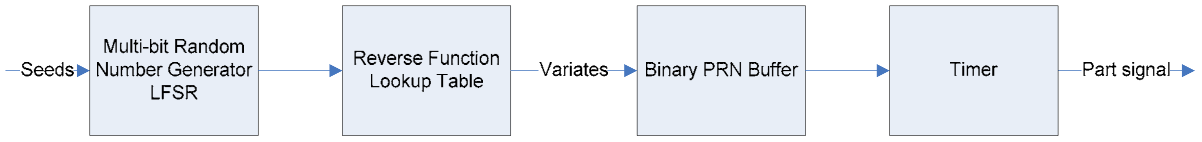

Seeds
Multi-bit Random Number Generator LFSR
Reverse Function Lookup Table
Variates
Binary PRN Buffer
Timer
Part signal

Figure 4.4: Random signal generator

The LFSR-based uniform random number generator is capable of producing one 16-bit binary random number in each cycle, and these numbers are fed into a memory-based look-up-table (LUT), where reverse function data are stored. The outputs of the LUT are non-uniform random numbers following specific distributions. Some of these numbers will be lost due to limited capacity of the buffer, however, this dose not matter since these numbers are following independent identical distribution (IID). The final outputs will be a time series of electronic pulse (or pulse group in the cases of multiple part types) with their interval following a certain probability distribution (Figure 4.5)

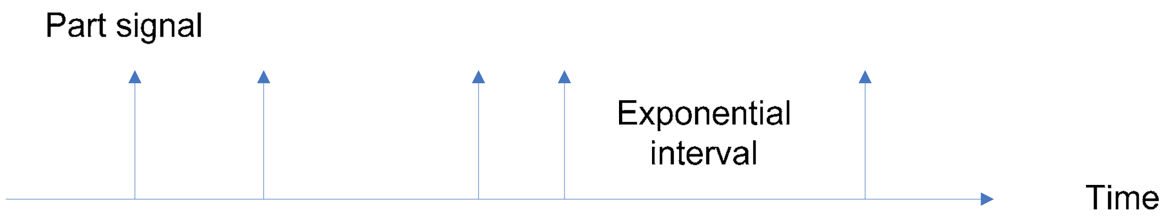


Figure 4.5: Part signals

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 occupies roughly 23% of the total execution time. Therefore, if a way can be found to reduce the time spent to generate the random numbers, the simulation speed can be dramatically improved. Following this initiative, a hardware random number generator based on LFSR is designed.

The heart of the random number generator is the LFSR. A LFSR has (2^N - 1) states, where N is the number of registers in the LFSR. The contents of the registers are shifted right by one position at each clock cycle. The feedback from predefined registers or taps to the left most register are XOR-ed together. LFSRs have several variables: the number of stages in the shift register, the number of taps in the feedback path, the position of each tap in the shift register stage and the initial starting condition of the shift register often referred to as the “FILL” state.
The maximal length sequence for a shift register of length N is $2^N – 1$. For example, an eight stage LFSR will have a set of m-sequences of length 255. There are many combinations of taps that produce small cross correlation m-sequences. Consequently, it is possible to define a set of taps that produce a collection of small cross correlation sequences for a constant length shift register. In this research, a widely used LFSR based random number generator, usually used for providing random sequences for the in-phase channel in CDMA modulation scheme, is adopted (Figure 4.6). The length 2^{17}-1of the random number sequence should be well enough for most simulation applications. While generating multi-bit random numbers, accumulation method is utilized. This is a straightforward extension of the one-bit methods. It utilizes k copies of identical one-bit generator hardware to generate k bits concurrently. The major advantage of this method is the operation speed, which is identical to that of a single-bit generator.

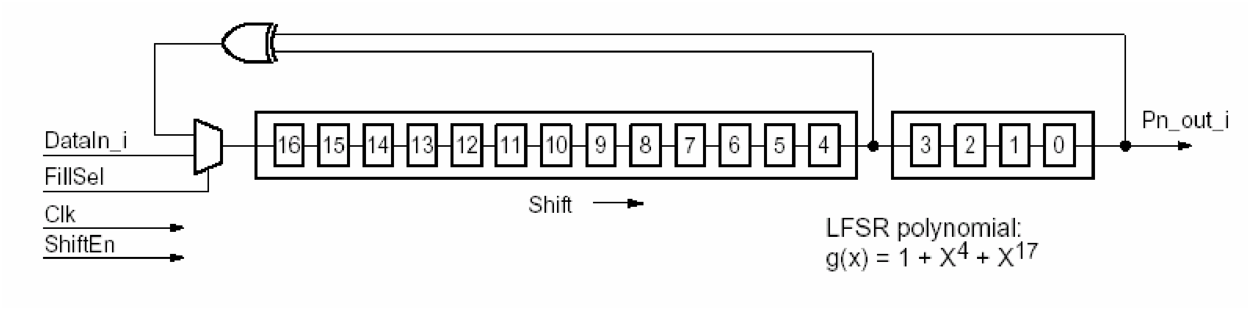


Figure 4.6: Linear Feedback Shift Register (LFSR)

From above discussion it is easy to find that this hardware simulator is driven by binary random numbers, instead of floating point random numbers used in traditional simulation. Here the goal is to correctly control simulation advancement, not to generate accurate random numbers, therefore a tradeoff between accuracy and cost should be obtained. The precision of a random number depends on its width (number of bits). The more bits it uses, the more accurate it becomes. However, since all those reverse function data have to be stored in memory-based LUT, the memory size and data bus width will determine the limit of the random number width. For example, if 16-bit random numbers are used, the required memory size should be at least 128K Bytes. How does precision of random numbers affect the simulation results? This is hard to determine by analytical methods, but can be found by experiments. On the other hand, lack of random number precision could also be offset by increasing the number of simulation replications.

4.4.2.2 Event processing unit

Event processing unit is the mapping of a machine center. It mimics machine behaviors like buffering, part selecting and processing. It also provides a bunch of signals which are used to describe current machine status. A typical scenario of the unit can be as follows: when a part signal arrives from outside, it is first stored in the buffer (in this case the buffer is realized by an FIFO queue, which indicates the first-in-first-out scheduling rule is selected). If the machine is currently empty and ready for processing parts, the part signal will be sent to part register. Meanwhile, a processing start signal tells the timer inside the processing core to begin counting, according to the stochastic processing times generated for each part. The operation index manager then increases the operation sequence number by one. When time is up, the processing core sends a process over signal to part register. Then the finished part is sent out. All these functional components are synchronized by one clock. Here one clock cycle corresponds to one simulation step, and any changes of the machine status can only happen at the end of a step. The complete simulation unit framework is shown in Figure 4.7.

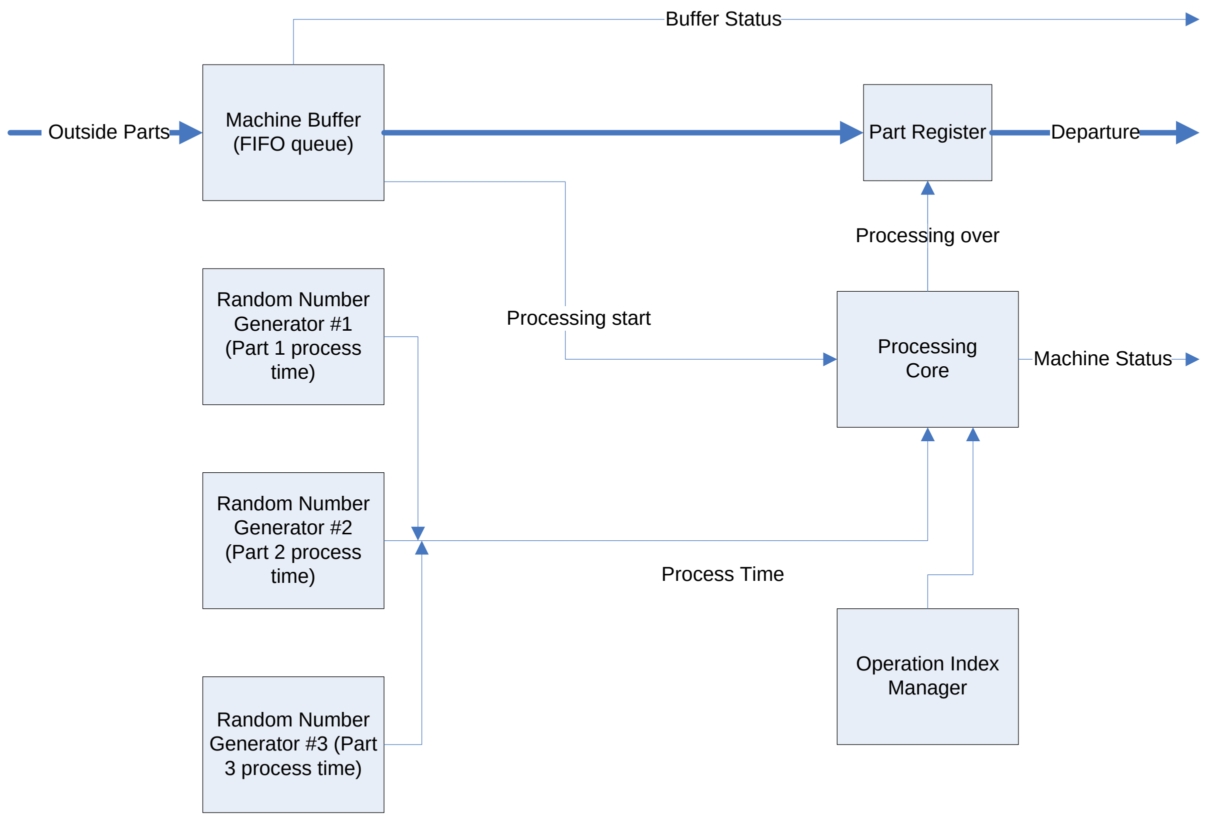

Buffer Status
Outside Parts
Machine Buffer (FIFO queue)
Part Register
Departure
Processing over
Random Number Generator #1 (Part 1 process time)
Processing start
Processing Core
Machine Status
Random Number Generator #2 (Part 2 process time)
Process Time
Operation Index Manager
Random Number Generator #3 (Part 3 process time)

Figure 4.7: Hardware simulation unit

In the above figure, Random number generators are used to generate processing times for different part type. The processing times may follow different distributions. Part register is used to temporarily store part information while the part is being processed. Processing core is the control unit.

4.4.3 Modeling manufacture system flexibility

FMS is featured by various flexibilities. In order to accurately grasp the system characteristics the simulation model must be able to realize those flexible features. In software simulation model, these can be easily implemented. For example, when parts arrive at a workstation, several rules are available for selecting next part to process. In this new hardware simulation modeling, although flexibility realization is not as easy as in software, limited flexibilities still can be realized, thanks to the hardware re-configurability feature.

4.4.3.1 Dispatching flexibility

Dispatching flexibility of the simulator means that it is capable of evaluating multiple dispatching rules. This function can be realized in two ways. The first way is to evaluate one rule at a time, just like its software counterpart. It works like this: At the beginning rule 1 is configured into simulator, and after the simulation run microprocessor read another configuration data, which corresponding to rule 2, and download the data into FPGA for another simulation run. The simulator continues to update configurations until all dispatching rules are evaluated and compared. This way requires minimum hardware resources, however, could not fully leverage the benefit of massive spatial parallelism that reconfigurable devices bring to us. The second way is to configure all dispatching rules into several copies of simulator. Then attach all these simulators to OPB bus in order to communicate with microprocessor (Figure 4.8). In this case multiple simulation processes are run consecutively without interfering with each other. After simulation run, microprocessor reads all results one by one, and compares them according to pre-defined criteria.

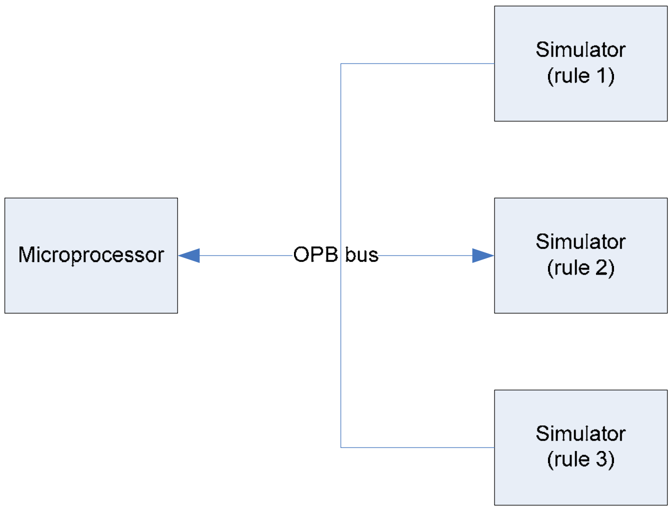


Figure 4.8: Dispatching flexibility

4.4.3.2 Routing flexibility

Routing flexibility means that after a part has been processed in one workstation, it can choose next destination from a set of available workstations according to some criteria. In hardware simulator model it is more difficult to realize than in software model. Because more routing choices mean more physical connections and more signal transmission interactions. In this research only a limited routing flexibility is realized. That is, the part can choose one of the two manufacturing cells when it is release to shop floor. For example, if a part is originally scheduled to be processed in cell 1, then it can change to cell 2 if there is a machine failure in cell 1.

CHAPTER 5 IMPLEMENTATION TECHNIQUES

5.1 High level modeling and prototyping

Emerging high-level hardware description and synthesis technologies along with FPGA devices have significantly lowered the threshold for hardware development. These technologies can be incorporated into a manageable and affordable prototyping framework for exploring and evaluating new microprocessor designs. This approach provides another middle ground between paper design and full scale implementation efforts where the designer can quickly test preliminary ideas at a very low cost. Although software simulators can produce convincing answers just like actual hardware, developing a prototype provides a concept proving platform and directly addresses questions in design complexity and feasibility. However, in order for prototyping to be an effective technique, the cost, risk and effort required should not greatly exceed those needed by simulations. By combining high-level hardware description technologies and FPGAs, researchers can prototype preliminary ideas without risking a great setback if the trial fails. The ability to quickly continue design refinements also allows for a more interactive design approach in finalizing the system architecture. As a supplement to simulation, an FPGA prototype is a powerful demonstration because it forces a designer to produce a complete and precise description of a design.

5.2 Introduction to FPGA prototyping platform

.

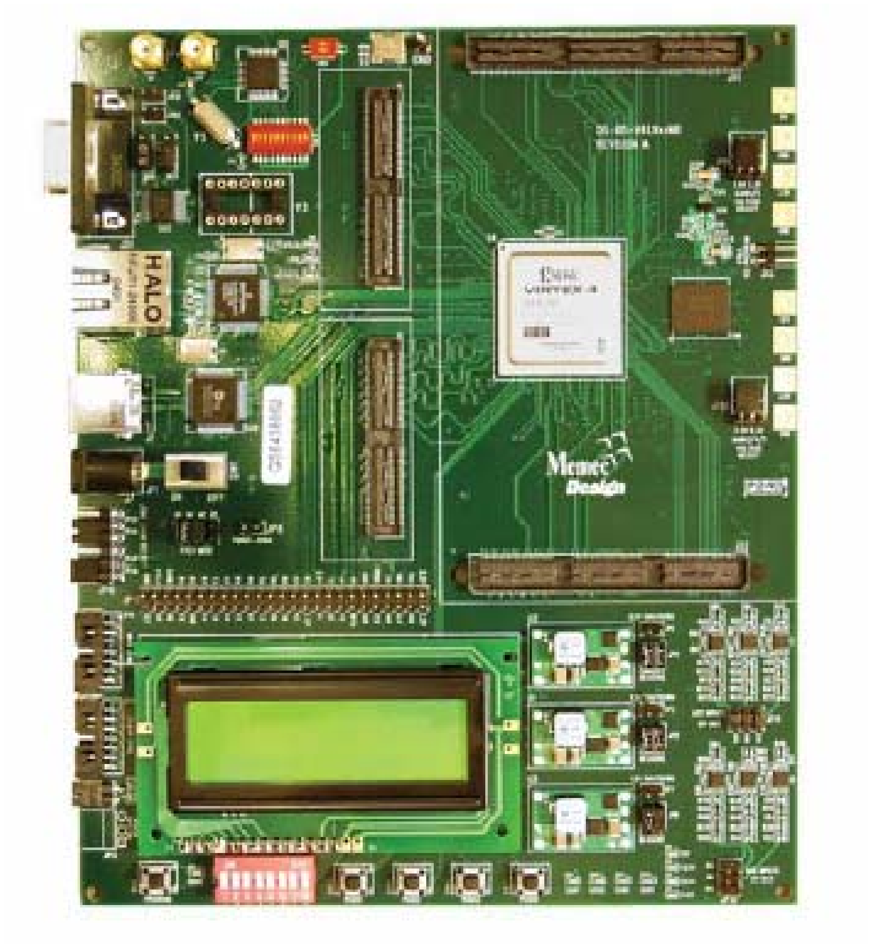


Figure 5.1: Virtex-II based development kit

The Virtex-4 MB Development Kit (Figure 5.1) offers an advanced development platform for the new Xilinx Virtex LX and DSP focused SX family of platform FPGAs. The MB board provides both high performance and practical prototype interfaces common to most designer's needs. The board is based on 4VLX60, which is the highest density FPGA chip commercially available at this time. The Virtex-4™ Family is the newest generation FPGA from Xilinx. The innovative advanced silicon modular block column (ASMBC) based architecture is unique in the programmable logic industry.

5.3 Development environment and techniques

The procedures of developing a FPGA based application is outlined in Figure 5.2.

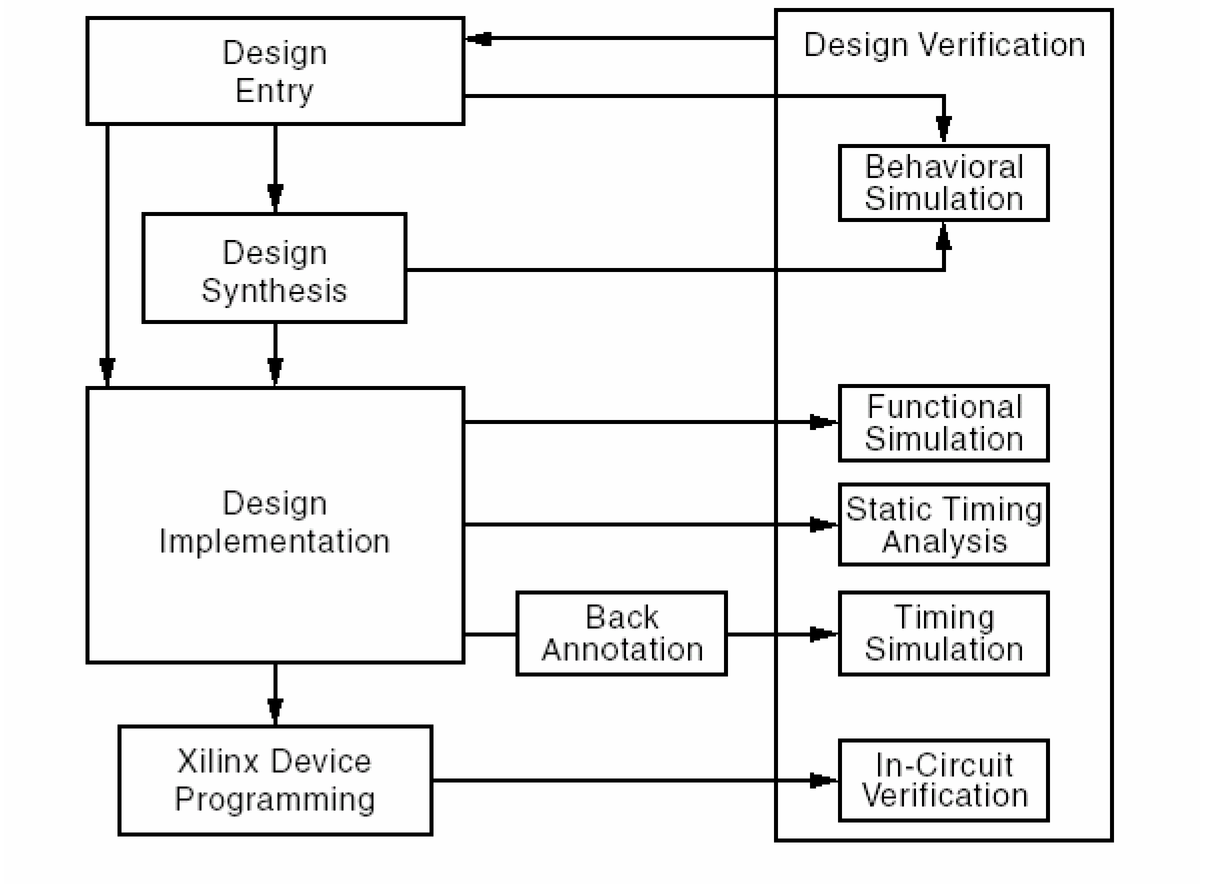


Figure 5.2: Design Flow

- Design Entry

The classic way of design entry for FPGAs is an interactive graphical schematic entry tool, which is to build logic networks by selecting library elements and connecting them manually. While the graphical design entry is good for glue logic, it is not appropriate for behavioral designs like state machines or complex designs. For this purpose, a textual specification is more adequate, which describes the design using a hardware description language (HDL). Recent design entry environments for FPGA applications usually support both schematic entry and HDL entry.

- Design simulation

Simulation is used to validate the design. In the FPGA design flow, there are typically two steps of simulation, functional and timing simulation. The functional simulation verifies that the specification meets the desired function of the circuit. It is typically done after the design entry by simulation software. At this time, the exact timing behavior of the circuit is not yet known, as this depends on the wire lengths, which are determined by the subsequent placement and routing steps. Thus, a timing simulation is typically done after placement and routing, using additional worst case delay times gained through back annotation from the final mapping information. This simulation models the behavior of the circuit more accurately than the functional simulation and can be used to verify timing constraints.

- Synthesis

Synthesis is the compilation of high-level behavioral and structural HDL statements into a flattened gate-level net-list, which then can be used directly either to lay out a printed circuit board, to fabricate a custom integrated circuit or to program a programmable logic device. As such, synthesis is quite similar to compiling a program in a conventional high-level language, such as C. The difference is that, instead of producing object code that runs on the same computer, synthesis produces a physical piece of hardware that implements the computation described by the HDL code.

- Implementation

Design Implementation is the process of translating, mapping, placing, routing, and generating a BIT file for the design. The Xilinx Foundation Series provides everything required to design a programmable logic device in an easy-to-use environment. This fully integrated tool set allows users to access design entry, synthesis, implementation and simulation tools in a ready-to-use package. Every step in the design process is accomplished using graphical tool bars, icons and pop-up menus supported by interactive tutorials and comprehensive on-line help. The Xilinx Foundation Series features support for standards based HDL design. HDL configurations include integrated VHDL synthesis with tutorials and graphical HDL design entry tools to turn new users into experts quickly and easily.

5.4 One machine simulation unit

Design and implementation of the one-machine emulator unit is the key of the whole system. It is the basic component of a functioning FMS simulator. Realization of the one-machine simulator will provide some useful information on whether the hardware simulator is achievable or not. It will also provide a rough estimate of speedup over software models. Preliminary validation process can also be done by comparing the results between hardware and software model.

5.4.1 Hardware random signal generator (PRN)

5.4.1.1 Previous research on hardware PRN

Generating random numbers using hardware has been studied for some times. A bit random number generator using a feedback shift register (FSR) was first introduced in 1965 [Tausworthe 1965]. It could generate arbitrarily long sequences of random numbers without the multidimensional non-uniformities found in commonly used LCG algorithm. Two years later Lewis and Payne [1973] refined Tausworthe's generator to create a generalized FSR (GFSR). In 1983, Barel [1983] presented a hardware random number generator to produce the Tausworthe sequence with generation time independent of the number of bits per word N. Non-uniform random numbers (variates) are difficult to generate for hardware, because this involves massive floating point computation. An alternative method is to store pre-generated distribution data in a look-up table (LUT) and us binary random numbers as index to find them. A large amount of memory will be needed for this method. As a trade-off, an efficient and scalable fixed point method for generating random numbers for any probability distribution function in a FPGA is developed [McCollum et al, 2003]. This method combines the features of LUT based generator and software LCG generator.

5.4.1.2 A special LUT-based binary PRN generator

In this project, the LUT-based non-uniform random number generation method is adopted. The first step of the PRN design is to decide the width of random numbers should be decided. That is, how many bits should be used to hold a random number? As previously stated, this is a tradeoff between accuracy and cost. Larger random number range can support smaller time-scaling factor, which implies a more accurate simulation but at bigger time cost. For example, if one clock cycle in simulation represents one second in real world (smaller factor), then the simulation is 60 times more accurate than the case in which one clock cycle represents one minute (larger factor). However, given a predefined scaling factor, larger random number range may not necessarily improve the simulation results. Since in this design simulation is a variation of event driven, the simulation clock is discrete and must be the integer multiple of a clock cycle. It is apparently most efficient to use one clock cycle to represent one simulation step. Under this circumstance, a random number width needs to be determined in order to reflect inter-arrival time properly. To illustrate this issue, see the example below:

Suppose we know the inter-arrival time of a part following exponential distribution with a mean of beta. Its accumulative probability function and inverse probability function are as following: F(x) = 1 – Exp(-x/Beta); x >= 0; Beta >0;

Z(p) = - Beta * ln (p) 0 <= p<1; Beta>0, where Beta is the scale parameter.

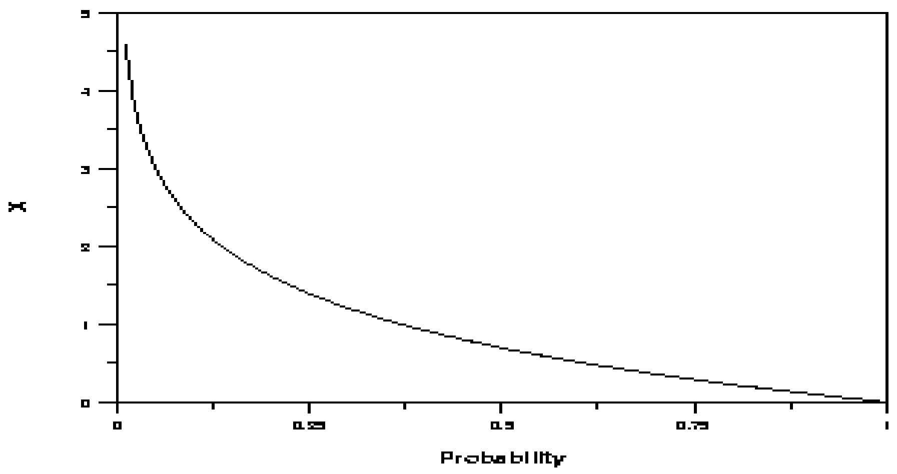


Figure 5.3: Exponential distribution

If the variate conversion covers all the random number range, then the width of random number should be infinite, therefore some kind of sacrifice must be made. Suppose the conversion is designed to cover all 16-bit numbers, then it should at least has a range of M, where

$M = -Beta * Ln(1/2^{16}) < 2^N$, N is the width of random numbers

For example, if the inter-arrival has a mean of 120 time units, then the width of random number should be at least 11 bits. Therefore, 16-bit random numbers are good enough for inverse function of exponential distribution.

Realization of the 16-bit binary random number generator is shown in schematic (refer to Appendix A). Sixteen LFSRs are grouped together to form the generator. Each of the LFST has a structure of $1+X^4+X^{17}$. Different initial values will be stored in those shift registers. One 16-bit uniform random number will be produced in each clock cycle. These numbers are first stored in a buffer of 256 units in depth, and then fetched by a timer to produce the part signal series.

5.4.1.3 Verification of PRN generator

To verify the part signal generator, some inter-arrival time data has been collected. The following numbers are obtained from the exponentially distributed output signal series (mean 45 clock cycles), representing distance between generated pulses.

23,75,16,3,30,22,33,54,16,43,60,92,153,
30,70,171,49,79,41,20,28,44,26,41,18,27,
68,40,121,23,28,29,119,59,69,9,55,47,78,
13,64,62,76,42,59,20,140,17,144,17,32,90
37,49,100,3,10,4,107,23,59,27,15,63,94
3,47,9

The data are fitted into distribution histogram (Figure 5.4), and it appears to match the expected distribution well.

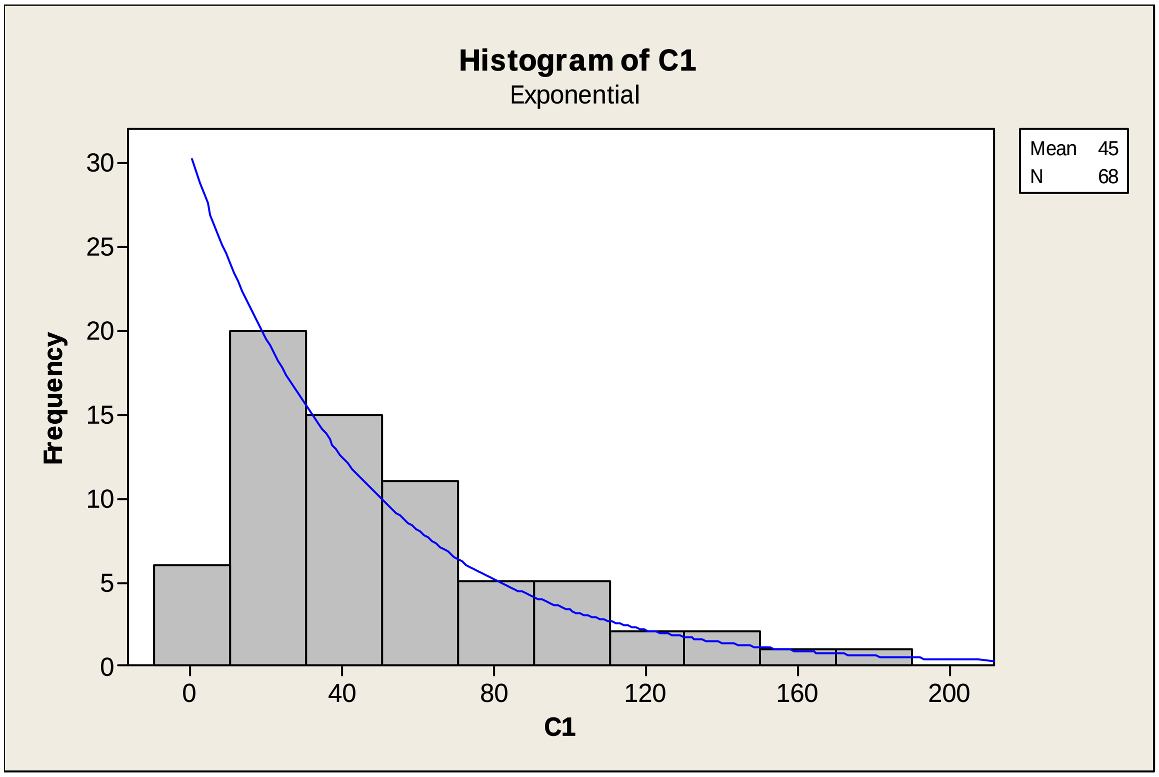


Figure 5.4: Sample histogram

The above procedure indicates the random variates are generated correctly. Next step is to make sure the variates can be correctly translated into time intervals. Logic simulation is done on a random signal generator. Results are shown below:

Signal distribution: Exponential (45)

Simulation duration: 20000 – 42 = 19958 clock cycles (Where 42 cycles are used by initialization process)

	1	2	3	4	5	6	7	8	9	10
Signals	401	422	435	426	424	419	450	449	422	437

Average signals generated: 428.5

95% Confidence interval: [417.958, 439.042]

Since the expected number of signals should be 19958/45 = 443.5

Therefore, it can be concluded that the signal generator is producing fewer signals than a Poisson process. This problem should be solved before proceeding to the next stage.

By carefully examining the signal waveforms, one may notice that due to hardware delay, the intervals between two signals are prolonged by 2 clock cycles. For examples, if two signals are expected to be one cycle away, the actual gap turns out to be 3 cycles. The smaller the mean value is, the more significant the affect would be. One possible solution is to deduct 2 from each value in variates stored in LUT. This deduction will be compensated by delay in hardware later.

After this adjustment, a group of data is collected as below:

Parameters are set to be the same as those in above experiment

	1	2	3	4	5	6	7	8	9	10
Signals	420	454	489	463	431	438	453	434	429	428

Average number of signals: 443.9

95% Confidence interval: [428.954, 458.846]

This set of data seems to match theoretical mean very well. Therefore, the function of part signal generator is verified.

5.4.2 Part signal structure

When there is only one part type being processed, an electronic pulse is enough to present the part flow. However, in case of multiple part types and different operation sequences, more complex signal structure should be constructed. Currently an 8-bit signal vector is used to represent a unique part status, with the higher four bits representing part type and lower four bits representing operation sequence. Therefore, the simulator, at its present form, can handle up to 7 different part types (“0000” is not valid) and 6 operation sequences (“0000” and “1111” are reserved).

5.4.3 Processing core and part buffer

The processing core controls the part flow inside the machine center. If it is not currently processing any part, it will check the part buffer status. If the buffer is empty, then the processing core remains idle. Otherwise, it will load the part into part register, obtain the part processing time from the time generator and begin timing. Meanwhile the machine status signal is set to busy. When time is up, it will instruct the operation sequence manager to add the part sequence number by one, set machine status to empty, and exit part from the machine. Once again here the hardware delay has to be considered. As an example, suppose processing time follows triangular distribution, and a value of 24 clock cycles is generated. But the actually processing time in the system will turn out to be 27 clock cycles. The hardware delay can be regarded as transportation delay or just set-up time. Usually it depends on the circuit design. Better designs should incur less delay in the system. Here only actual processing time is used for calculating machine utilization, so compensation is made by subtracting 3 from processing time stored in LUT.

The part buffer is where the scheduling rule is being realized. Currently it is modeled by a FIFO queue which can store up to 64 parts. If the buffer is full, then a system error message will become valid. (Refer to Appendix A and B for schematics and output waveforms.)

5.4.4 Downtime control and statistics collection

Machine downtime control will be modeled by microprocessor. If the “done” control signal is received by emulator unit, then it will disable system clock signal, thus stop all activities happening inside the machine. The status will be reserved and recovered when “done” signal becomes invalid later (refer to appendix B for waveform illustration).

The emulator unit provides a number of system signals, which can be used for statistics collection. The “throughput” signal shows a binary number of parts being processed; “busy” signal can be used to find out machine utilization by calculating the percentage of aggregate busy time to overall simulation time; “buffer count” signal can be used to calculate part average waiting time in queue. Data will be collected by microprocessor after each simulation run.

Figure 5.5 shows the scenario of single part type. “Part in” signal is a time series of pulses with interval following exponential distribution. “Part out” signal indicates finished parts leaving the machine. “Busy” signal indicates machine is in production mode. It is composed of a series of square waveform pulses, with their width equivalent to part processing time.

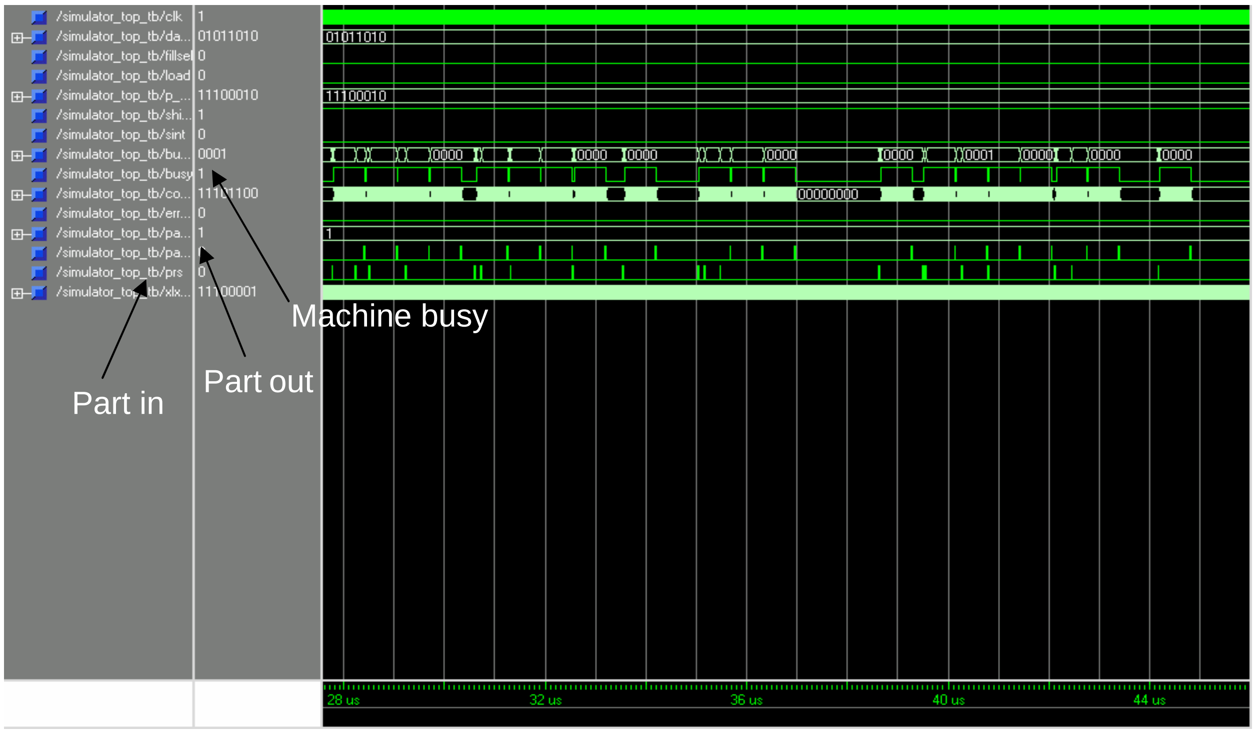

/simulator_top_tb/clk 1
/simulator_top_tb/da... 01011010
/simulator_top_tb/fillsel 0
/simulator_top_tb/load 0
/simulator_top_tb/p_... 11100010
/simulator_top_tb/shi... 1
/simulator_top_tb/sint 0
/simulator_top_tb/bu... 0001
/simulator_top_tb/busy 1
/simulator_top_tb/co... 11101100
/simulator_top_tb/err... 0
/simulator_top_tb/pa... 1
/simulator_top_tb/pa... 1
/simulator_top_tb/prs 0
/simulator_top_tb/xlx... 11100001
Machine busy
Part out
Part in
28 us
32 us
36 us
40 us
44 us

Figure 5.5: Input/Output waveform

5.5 Two-Machine cell implementation

A simple cell with two machine centers, one lathe and one milling machine, is built into the simulator. This is the basic simulation model taking consideration of interactions between two machine centers. If the two-machine emulator can be realized and verified, then it will provide good insights on two mostly concerned issues: “How much speedup can this simulator achieve?” and “Is this hardware simulator a valid representation of the FMS?”

5.5.1 Two-Machine cell simulator framework

The two-machine model consists of a part dispatcher, a lathe, a mill and an exit station. Multiple part types can be processed, with processing time following different triangular distributions for each machine.

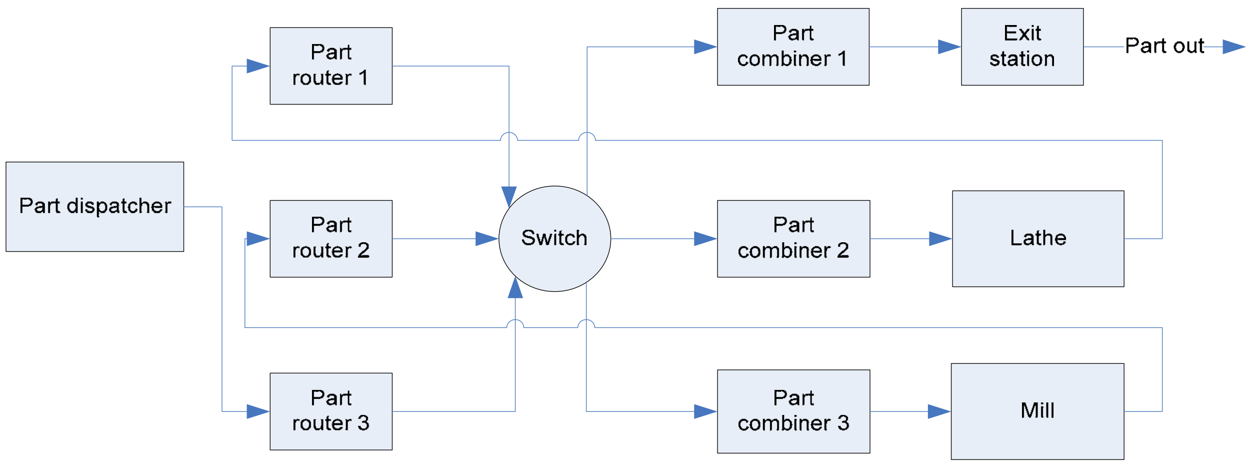


Figure 5.6: Two-Machine cell

The system is designed to be bus-based architecture in order to achieve scalability. As the number of machines in the FMS becomes larger, the bus will become a bottleneck of the system if it cannot increase traffic handling ability accordingly. The idea is to increase bus width while new machines are added in, just like adding more lanes to a crowded highway. A special switch network will be used to decide the connection between machines and bus, based on the part sequences. The connection is to be reconfigured each time there is a new process plan. Parts flowing on different lanes of the bus may finally merge into one exit, such as a lathe or a mill, so there must be a combiner to arrange the exiting order. The merging points should be carefully chosen to avoid heavy traffic. Usually they are located at the entrance of each machine center. To apply maximum parallelism, each lane of the bus has its own router. As a result, the special bus of this system can be regarded as consisting of all the routers, switch network and part combiners (Figure 5.6). (Refer to Appendix A and Appendix B for the schematic and output waveforms.)

5.5.2 Part Dispatcher

Part Dispatcher consists of a number of part signal generators and part information registers. Each part signal generator emulates the arrival pattern of one part. The part signal generator produces arrival signals (pulses) following different exponential distributions, and then the signals trigger part information register to send part signal (8-bit vector) out. The part signals usually contain part type information and an initial operation sequence number. When multiple part signals are generated at the same time, they are merged into one exit via part combiner.

5.5.3 Part Router

Part Router determines where the part should go next, based on its operation sequence. To quickly index the part signal to its sequence information, content addressable memory (CAM) is introduced. A Content Addressable Memory is a storage array designed to quickly find the location of a particular stored value. By comparing the input against the data in memory,
CAM determines if an input value matches one or more values stored in the array. If the comparison is done simultaneously, the CAM is said to be performing at maximum efficiency. If a match exists, it is found in a single clock. Part information will be stored in CAM, and if a part signal matches the content, then the active address will be decoded, according to process plan, and used to control a gateway which branches into different destinations. (Refer to Appendix A and B for schematics and IO waveforms.)

5.5.4 Part Combiner

Surprisingly, this is one of the most difficult system components to design. Since there can be an unknown number of parts arriving at the merging point in the same clock cycle, the task is to arrange them properly and send them out sequentially. While dealing with these concurrently arriving parts, all successive arrivals should be stored in a buffer for later transactions. Any part missing is not allowed. Ideally, a circuit should be designed to fulfill the function depicted in Figure 5.7.

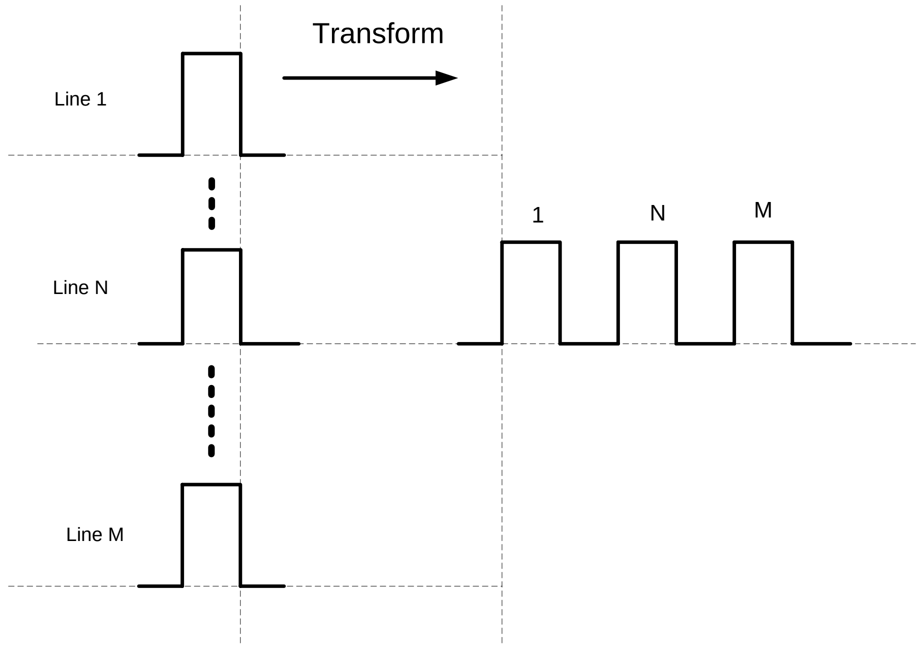


Figure 5.7: Ideal part combiner function

This is easy to realize this function using hardware description language, such as VHDL. However, it would be very difficult to translate into real circuit without incurring much delay. To bypass this problem, a part combiner with only two inputs, namely Part_2to1, is designed with a transmission delay of 3 clock cycles (refer to Appendix A and Appendix B for schematic and waveforms). More concurrent inputs can be realized by concatenating the part combiners, at a cost of longer delays. Concurrent parts will be arranged randomly and sent out one by one without priority. Too many concurrent parts may cause an error on buffer overflow.

5.6 Four-Machine system implementation

5.6.1 Robot model design

Robot model is the most complicated model in the design. There are two manufacturing cells in the desktop FMS. Each cell consists of one lathe, one mill and one robot. Both cells share the part dispatcher and exit station (Figure 5.8). Robot interfaces with each machine center in the cell and receive parts from dispatcher, lathe and mill. If robot is busy, then the source module will be blocked. If there are multiple parts requesting robot at the same time, tie-breaking rule is a pre-defined order (for example, dispatcher-lathe-mill).

After the robot gets a batch of part signals, it stores them into a buffer and block non-empty source module. Then the signals are read out sequentially and processed by transporter. Each time when a part signal is loaded onto transporter, the resource module where the part comes from will be unblocked. The transporter will send parts to their destinations (lathe, mill or exit station). (Refer to Appendix A and Appendix B for schematics and IO waveforms.)

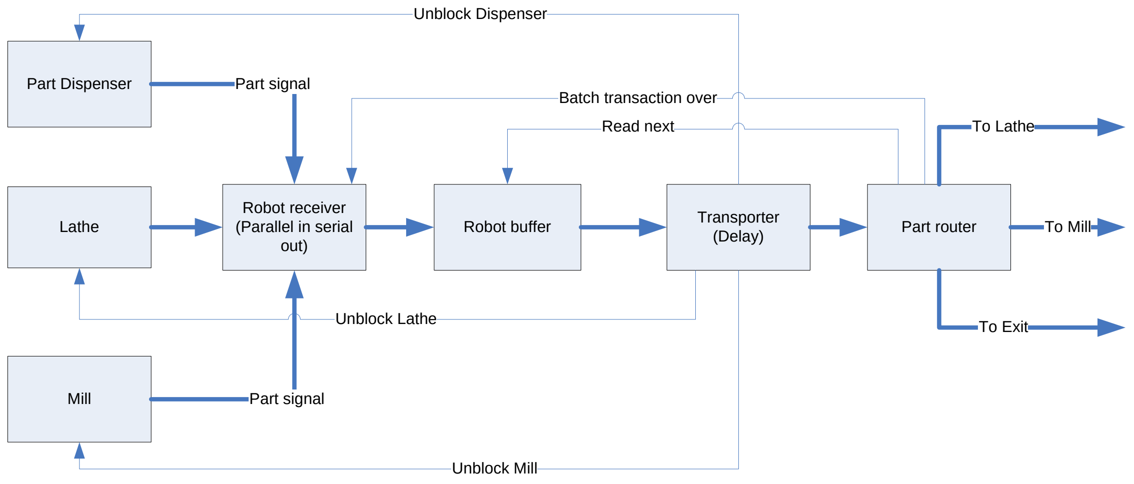


Figure 5.8: Robot framework

5.6.1.1 Robot Receiver

Robot Receiver is the input function module of a robot. Its main function is to convert multiple input part signal flows to one output signal flow (Figure 5.9). Whenever a part signal (or multiple part signals) arrives and robot is ready, the receiver will read a batch of signals from all input ports simultaneously, then perform a parallel-in-serial-out function and send all signals out sequentially. (Refer to Appendix A and Appendix B for schematics and IO waveforms.)

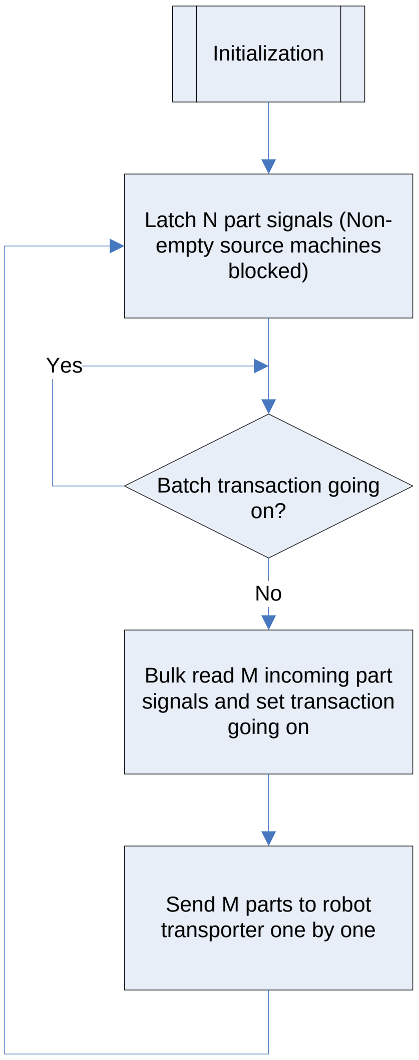

Initialization
Latch N part signals (Non-empty source machines blocked)
Yes
Batch transaction going on?
No
Bulk read M incoming part signals and set transaction going on
Send M parts to robot transporter one by one

Figure 5.9: Robot Receiver function flow chart

5.6.1.2 Transfer Time Generator

Transfer Time Generator calculates transfer time according to current robot position, part loading position and unloading position. Since all modules in a cell are placed in a straight line manner and the distances between any two of them are supposed to be equal, it is easy to find how long the transportation will last. For example, if the robot is currently at mill position and need to send a part at dispatcher to lathe, then it will need 6 minutes to finish the job (2 minutes for robot to go to dispatcher, 3 minutes for loading and one minute for traveling to lathe). (Refer to Appendix A and Appendix B for schematics and IO waveforms.)

5.6.1.3 Robot Transporter

Robot Transporter is a delay module, which is used to mimic the part transportation process (Figure 5.10). Each time it process one batch of part signals which arrived at the robot simultaneously. For example, at time t, two parts arrived at robot --- part A from part dispatcher and part B from lathe. Since the tie-breaking rule dictates part A has the priority, if robot is ready then it will begin to transport part A. Meanwhile, a signal is sent out to unblock the part dispatcher. After part A is processed, part B transportation will begin and a signal is sent out to unblock the lathe. When both parts are transported, the robot is set as “batch over” status and ready to read a new batch of part signals. (Refer to Appendix A and Appendix B for schematics and IO waveforms.)

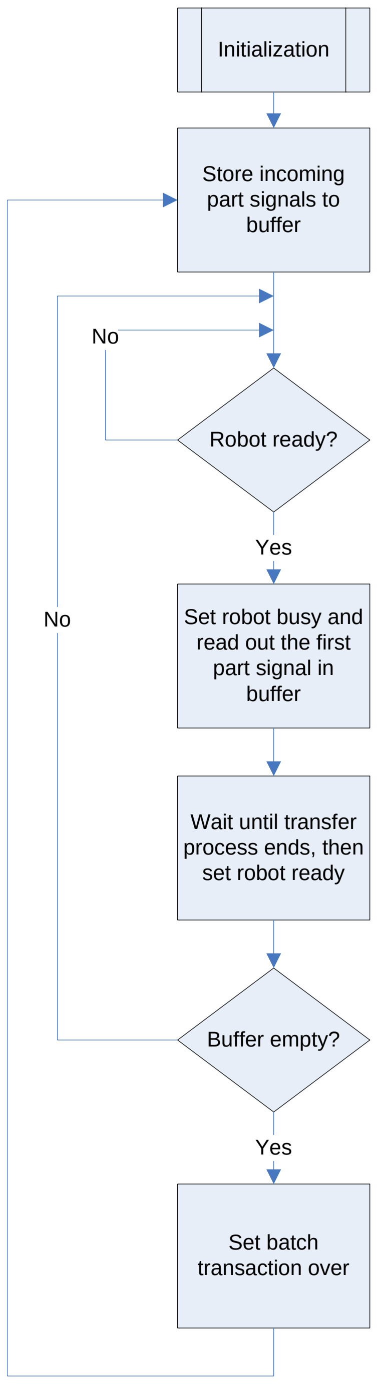

Initialization
Store incoming part signals to buffer
No
Robot ready?
Yes
Set robot busy and read out the first part signal in buffer
No
Wait until transfer process ends, then set robot ready
Buffer empty?
Yes
Set batch transaction over

Figure 5.10: Robot Transporter function flow chart

5.6.2 Complete Four-Machine FMS

5.6.2.1 Part Dispenser with buffer

Part Dispenser consists of a number of part signal generators and buffers (Figure 5.11). For each part type, there is a signal generator. After a part is generated, the dispenser will request a robot to pick it up. If the robot is busy, then the part is kept in buffer until an unblocking signal arrives.

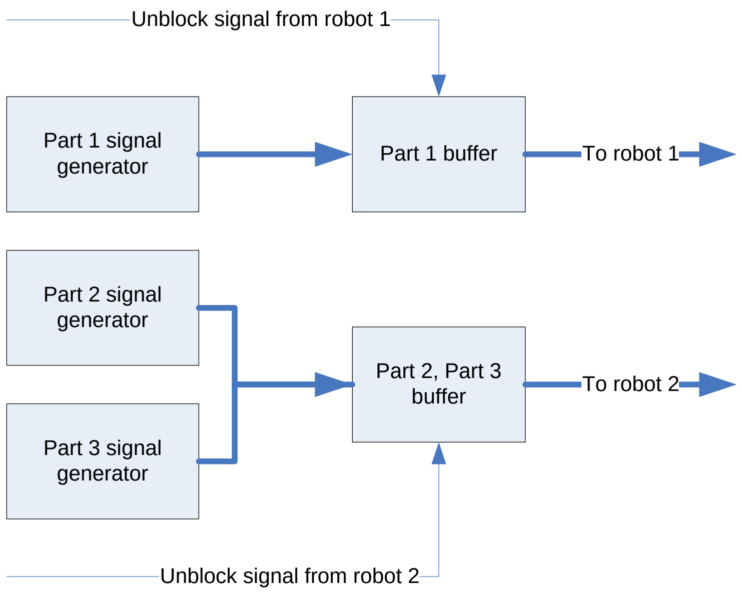


Figure 5.11: Part Dispenser with buffer

5.6.2.2 Complete four-machine FMS model

The final FMS model consists of one Part Dispenser, two manufacturing cell, one Exit station and a data acquisition module. (Refer to Appendix A and B for schematic and IO waveform)

5.7 Simulator prototype realization

In above sections the simulator design methodology has been described, and logical verification and validation have been performed. Till now the design stage is completed. In this section, a physical simulator prototype will be implemented into FPGA platform. The simulator will be connected to a microprocessor which provides operational interface.

5.7.1 Connecting to microprocessor

5.7.1.1 Introduction to MicroBlaze soft core processor and its developing environment

The MicroBlaze core is a 32-bit Harvard RISC architecture with a rich instruction set optimized for embedded applications [Xilinx 2003]. The processor is a soft core, meaning that it is implemented using general logic primitives rather than a hard, dedicated block in the FPGA. The MicroBlaze solution is designed to be flexible, giving the user control of a number of features such as the cache sizes, interfaces, and execution units. The configurability allows the user to trade-off features for size, in order to achieve the necessary performance for the target application at the lowest possible cost point (Figure 5.12).

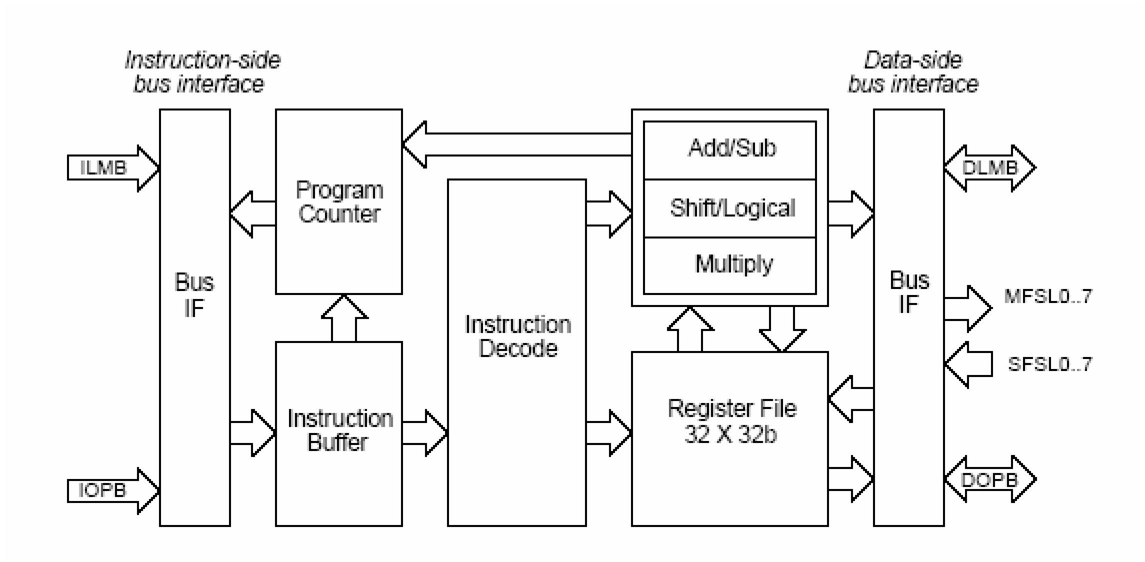


Figure 5.12: MicroBlaze core block diagram by Xilinx [2003]

MicroBlaze is especially developed for the advanced FPGA architecture. The 32 by 32-bit registers are lookup table (LUT) RAM based. It guarantees a very short register access time. For memory, either the on-chip block RAM or off-chip memory can be used. The access time to the on-chip block RAM is minimal because there are dedicated routing resources to access them. Due to the fact that MicroBlaze is using the available FPGA resources very efficiently, it is possible to clock MicroBlaze up to 150 MHz. Thus, up to 125 Dhrystone MIPS can be reached.

It is consequently the industry‘s fastest SCP for FPGAs. The MicroBlaze SCP can be customized for any application. Its barrel shifter, divide unit, data cache, instruction cache, and the FSL bus system are optional. The sizes of the caches are configurable from 2 to 64 Kbytes. Standard peripherals are provided as well and are Core Connect compatible. Consequently, they can be integrated in an embedded design very easy.

5.7.1.2 Xilinx Embedded Development Kit (EDK)

The EDK is a complete embedded development solution that includes a library of peripheral IP cores, the award-winning Xilinx Platform Studio tool suite for intuitive hardware system creation, a Built-On Eclipse software development environment, GNU compiler, debugger and more [Xilinx 2005 (2)]. The MicroBlaze processor is also supported by third party development tools and Real Time Operating Systems (RTOS). Figure 5.13 depicts the embedded software tool architecture. Multiple tools based on a common framework allow developers to design the complete embedded system. The system design consists of the creation of the hardware and software components of the embedded processor system, and optionally a verification or simulation component, as described below: The hardware component consists of an automatically generated hardware platform that can be optionally extended to include other hardware functionality that you specify. The software component of the design consists of the software platform generated by the tools, along with user-designed application software. The verification component consists of automatically generated simulation models targeted to a specific simulator, based on the hardware and software components.

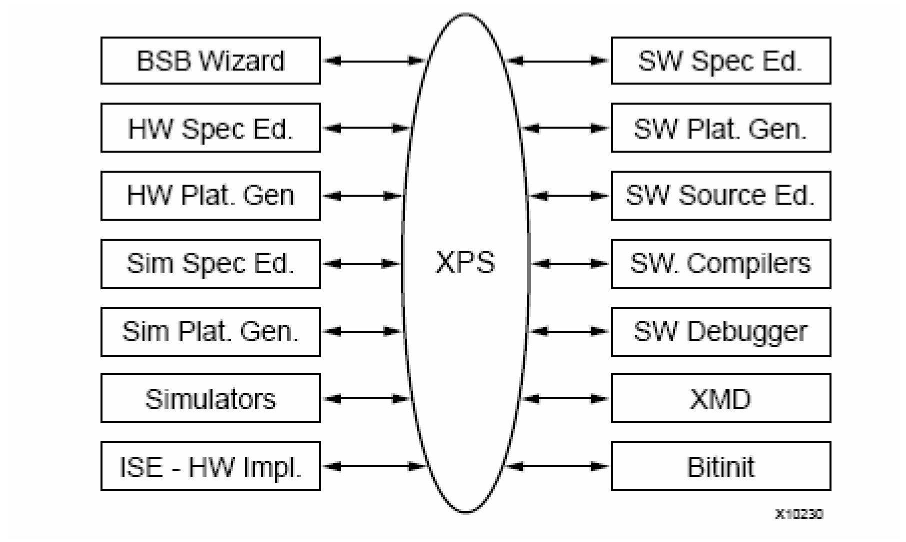


Figure 5.13: Embedded Software Tool Architecture

A typical embedded system design project involves the following phases: hardware platform creation; hardware platform verification using simulation; software platform creation; software application creation and software verification using debugging. Xilinx provides tools to assist in all the above design phases. These tools work with other third-party tools such as simulators and text editors that can be used by the designers.

5.7.1.3 Connecting simulator to microprocessor through custom IP integration

One advantage of a Soft Core Processor (SCP) is its flexibility: it uses only the processor features required for a specific application. Another advantage is its ability to integrate customized user Intellectual Property (IP) cores, which can result in a dramatic acceleration in software execution time due to algorithms being executed in parallel in hardware and not sequentially in software. Generally, there are two ways to integrate a customized IP core into a MicroBlaze-based embedded soft processor system. One way is to connect the IP on the On-chip Peripheral Bus (OPB) [Xilinx 2005 (1)]. The OPB is part of the IBM Core Connect on-chip bus standard. The second way is to connect the user IP to the MicroBlaze dedicated Fast Simplex Link (FSL) bus system [Xilinx 2004 (1)]. If the application is time-critical, the user IP should be connected to the FSL bus system; otherwise, it can be connected as a slave or master on the OPB. Since the microprocessor actually does not take part in the simulation process, high speed interaction with the hardware simulator is not urgent. Here the OPB connection method is selected due to its simplicity and versatility.

Since OPB bus is a variant of the powerful IBM Core Connect bus, it is rather difficult for non-professionals to build any applications. Fortunately, EDK uses Intellectual-Property Interface (IPIF) library to implement common functionality among various processor peripherals. It also gives you a set of simplified bus protocol called IP Interconnect (IPIC) which is much easier to use rather than operate on OPB bus protocol directly. Using the IPIF module with proper parameterization greatly reduce the design and test effort (Figure 5.14).

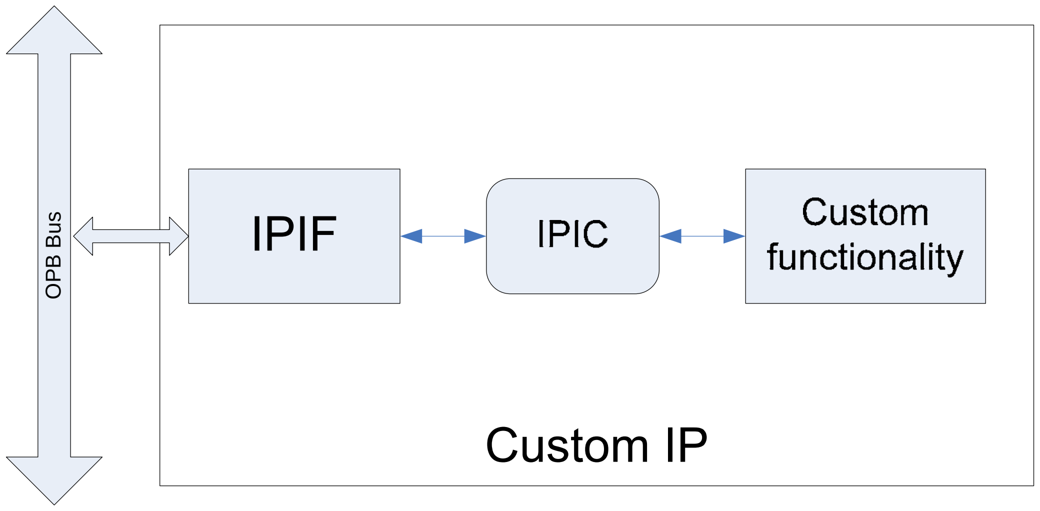


Figure 5.14: Using IPIF module in connecting peripheral

The first step of building the connection is to determine Interface: Identify the bus interface (OPB or PLB) custom peripheral should implement, so that it can be attached to that bus in the processor system. In this case, PLB is used for MicroBlaze to access main procedure from memory. OPB is the nature choice. The second step is to implement and verify the custom functionality, which has been done logically in previous research. Here the custom IP refers to the simulator itself. The last step is to import the custom IP to EDK by copying the peripheral to an EDK recognizable directory structure and creating the interface files so that other EDK tools can access the peripheral. The Create and Import Peripheral Wizard greatly simplify the custom peripheral creation process by guiding users through the design flow.

All previous schematic based designs cannot be connected to microprocessor directly, because only HDL based designs fit. Therefore, a HDL shell must be provided to encompass the simulator core, providing the same interface to the outside. Relevant procedures can be found in Appendix C.

5.7.1.4 Bus Functional Model (BFM) simulation

After implementing the core, a BFM simulation can be run to verify its functionality. BFM is much faster and easier to use than a manual test bench or system level simulation in XPS. BFM is a toolkit developed by IBM and integrated into EDK to help user quickly perform behavioral simulation on the peripheral itself. The wizard is able to generate a BFM simulation test platform, which includes a simulation system including the peripheral, an IP test bench which allow one to define user I/O stimulus, a sample script synchronized with the IP test bench to verify some basic features of the peripheral, as well as some other auxiliary files/scripts to make the BFM simulation an easy task.

5.7.1.5 System implementation

The whole system is built with Base System Builder (as shown in Figure 5.15). MicroBlaze processor connects to local memory via high-speed LMB bus. Simulation top level procedure is stored in on-chip block ram (BRAM). Hardware simulator is attached to OPB bus. During initialization MicroBlaze run top level program and generate control data such as random number seeds. Then the simulation subroutine is called and those control data are passed to hardware simulator to start simulation run. After finishing each replication, hardware simulator will generate an interrupt request and send the results back to microprocessor. When simulation process is finished, microprocessor will compute necessary statistics and generate a report. The report the will be sent to hyper-terminal on a PC through RS232 serial bus, which is controlled by OPB UARTLITE (a simplified serial communication protocol supported by MicroBlaze).

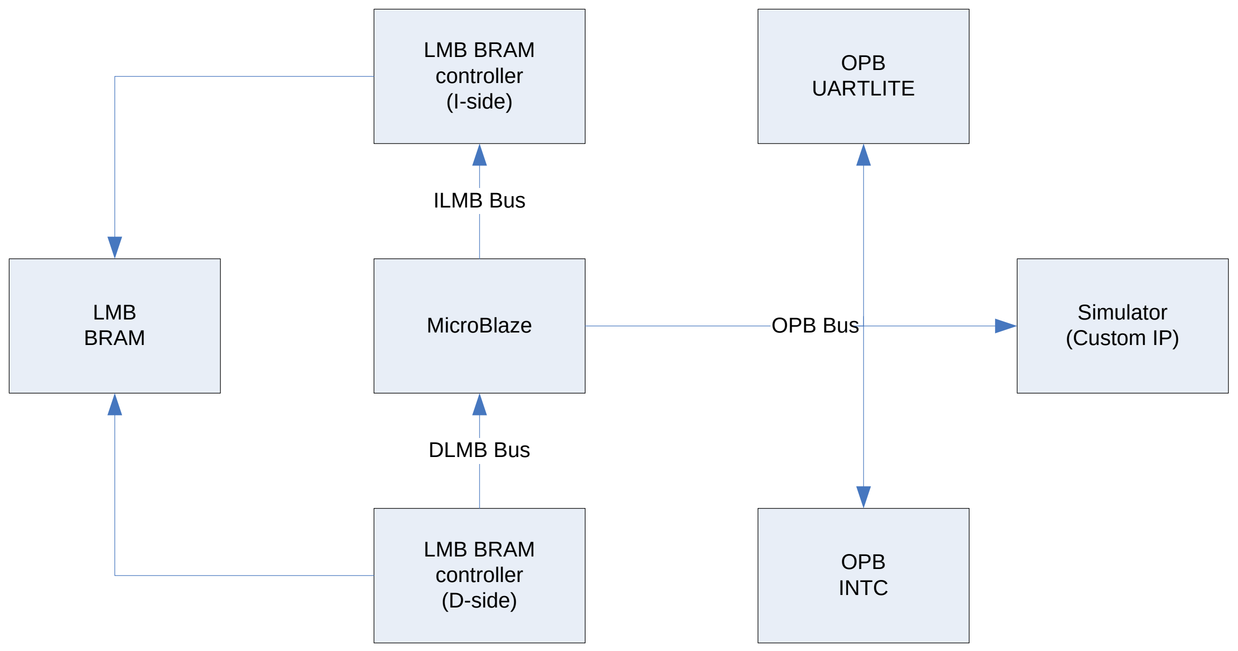


Figure 5.15: Microprocessor system architecture

5.7.2 Top level simulation procedure

The reconfigurable simulator realization is a hardware/software co-development process. The software application is written using C language. Figure 6.4 shows the top level procedure flowchart. When the configuration file is downloaded into FPGA, the microprocessor and simulator hardware are implemented. The C program is stored in the on-chip block RAM. Top-level program automatically begins to run the initialization process. Then the software generates random number seeds and control signal. Next it will call simulation subroutine to pass these data to simulator via OPB bus. As soon as the hardware simulator receives these data, it will begin the first replication of simulation process. When the replication is complete, the simulator will generate an interrupt and send simulation results back to microprocessor. Then new data are passed from microprocessor to hardware simulation and second replication starts. The process will repeat until desired replications have been carried out. Finally the microprocessor will collect all simulation results and compute statistics.

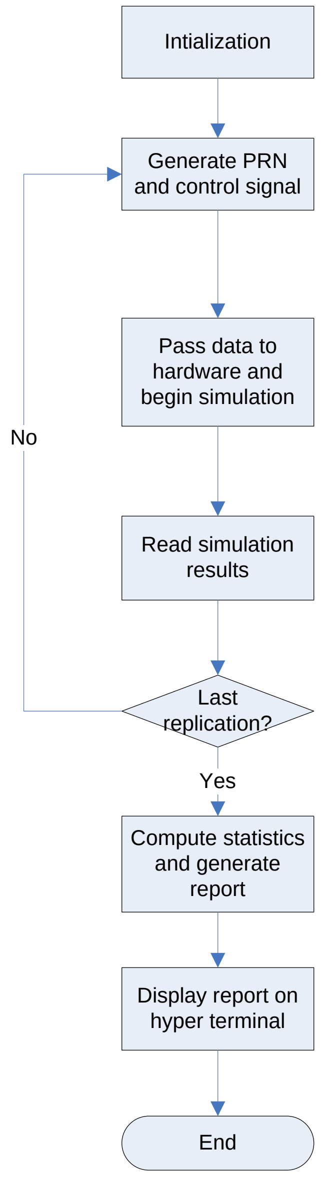

Intialization
Generate PRN and control signal
Pass data to hardware and begin simulation
Read simulation results
Last replication?
No
Yes
Compute statistics and generate report
Display report on hyper terminal
End

Figure 5.16: Main simulation procedure

CHAPTER 6 EXPERIMENTS AND RESULT ANALYSIS

6.1 Simulation Design

6.1.1 Determining values to be measured

For describing a manufacturing system, there are many aspects of a manufacturing system that we can measure. Examples include:

- Utilization - this is the percentage of the amount of time a resource is used.
- Mean Values - this gives an average of some measured quantity.
- Counts - this is just plain counting of some quantity in the system.

In a manufacturing system model, utilization of various machines is an important indication of system efficiency; mean values can be used to find potential bottlenecks in the system, or it can be used in measuring waiting times of jobs in the system, and consequently the queue length too. Using counts can give us an indication of the throughput of a system, thus one could count the number of goods produced in a day, and use this statistic to find out what the maximum rate of production the system can generate in one day. Among them, the machine utilization and system throughput will be considered first in this project.

6.1.2 Simulation time frame

Simulation can be classified as either terminating or steady state, depending on the goal of the designer. A terminating simulation is one in which the model has a specific starting and stopping conditions as a reflection of how the real system operates. A steady state simulation, on the other hand, is one in which the parameters to be estimated are defined in a long run. The initial conditions in this kind of simulation do not matter much, and the simulation run usually lasts for a considerably long time. In this project, since the aim is to compare the functionality of two systems, it is preferred to include transient period in the study. Therefore, the terminating simulation frame is chosen. With a terminating simulation, it is conceptually simple to collect the appropriate data for statistical analysis --- just by making certain number (N) of independent replications.

6.1.3 Simulation parameters

Suppose each minute in real world is represented as one clock cycle in simulation model, and a production period of about 2 weeks (7 days per week, 8 hours per day) is to be simulated. Then the simulation run length is set at 7000 clock cycles. Part arrivals are set to follow exponential distribution with different means, and processing times follow different triangular distributions. At first, 10 replications are used. Suitable replication number will be determined after some data analysis.

6.2 One-Machine unit simulation study

6.2.1 One-Machine simulation unit results

Terminating condition: simulation clock = 200000 ns

Part arrival: Exponential (45)

Process time: Triangular (15, 20, 25)

Replication time: 19958 clock cycles

Number of replications: 10

Simulation results are listed in Table 6.1

	1	2	3	4	5	6	7	8	9	10
T.P.	468	409	476	449	457	452	406	444	434	416
Busy (cycles)	9346	8186	9516	9047	9180	9075	8034	8829	8654	8246
Util.	0.4683	0.4102	0.4768	0.4533	0.4600	0.4547	0.4025	0.4424	0.4336	0.4132

Table 6.1: One-Machine unit hardware simulator results

T.P. stands for throughput.

Busy stands for total machine busy time (clock cycles).

Util. stands for machine utilization.

Statistics:

Average throughput: 441.1

Standard deviation: 24.33

95% confidence interval: [423.696, 458.504]

Average utilization: 0.4415

Standard deviation: 0.0258

95% confidence interval: [0.4231, 0.4599]

A comparison between results from hardware simulator and Arena is provided in section 6.4

6.2.2 One-Machine unit software model study

A single machine center model is built on Arena (Figure 6.1).

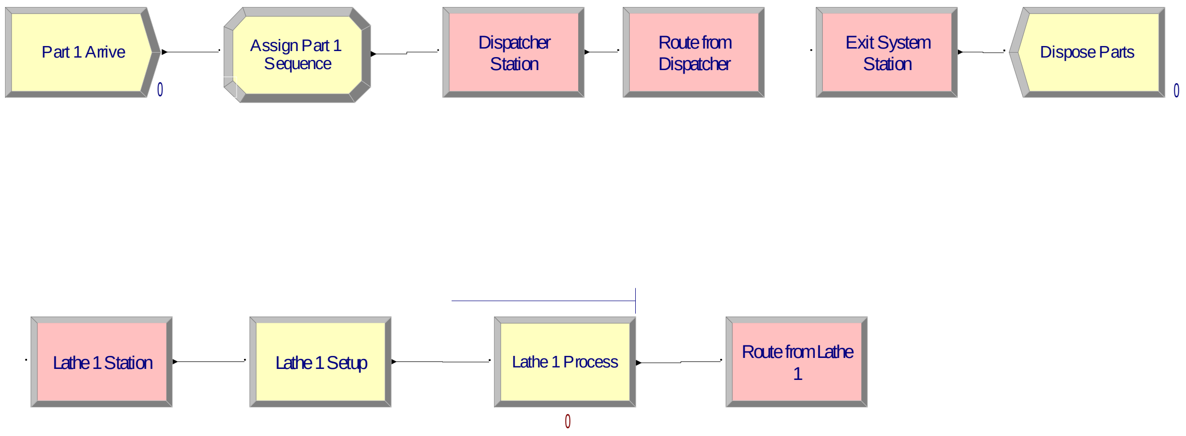


Figure 6.1: One-Machine Arena model

Simulation runs 10 replications

Terminating condition: simulation time = 200000 ns

Results are shown in Table 6.2:

	1	2	3	4	5	6	7	8	9	10
Part_ In	444	423	437	393	469	430	432	462	451	461
T.P.	444	423	436	393	468	429	432	460	451	460
Util.	0.4465	0.4248	0.4405	0.3957	0.4646	0.4295	0.4309	0.4596	0.4541	0.4595

Table 6.2: One-Machine ARENA model simulation results

Part_In: total number of parts coming into the system. Because there are usually some parts remaining in the system when simulation time is up, Part_In value is usually less than throughput.

Average throughput: 439.6

Standard deviation: 22.16

95% confidence interval: [423.750, 455.450]

Average utilization: 0.4406

Standard deviation: 0.0211

95% confidence interval [0.425500, 0.455640]

6.2.3 Comparison of One-Machine models

Table 6.3 lists collected data from two models:

	1	2	3	4	5	6	7	8	9	10
TP_H	468	409	476	449	457	452	406	444	434	416
TP_A	444	423	436	393	468	429	432	460	451	460
Util_H	0.4683	0.4102	0.4768	0.4533	0.4600	0.4547	0.4025	0.4424	0.4336	0.4132
Util_A	0.4465	0.4248	0.4405	0.3957	0.4646	0.4295	0.4309	0.4596	0.4541	0.4595

Table 6.3: One-Machine simulator statistics

TP_H: throughput from hardware simulator

TP_A: throughput from Arena model

Util_H: machine utilization in hardware simulator

Util_A: machine utilization in Arena model

Comparison is based on independent simulation runs, which means that two sets of random numbers are independent. F-test and T-test are used in comparing these two sample variance and means. The significant value for T-test is set at 0.05. Since F-test is two-sided, the significant value should be 0.025 for consistency. Variance reduction techniques such as common random numbers (CRN) technique might be needed later if results do not meet expectation.

1) For throughput:

F-test: H_0: there is no difference between two variances

H_1: variances of two outputs are different

Observed F = $S_1^2/S_2^2 = 24.33^2/22.16^2 = 1.205$

Critical F (9, 9) at alpha (0.025) = 4.026

Since F < F', the null hypothesis is accepted.

T-test:

H_0: there is no difference between two means

H_1: means of two outputs are different

t = (441.10 – 439.6) / SQRT ($24.33^2/10 + 22.16^2/10$) = 0.144

Here SQRT stands for square root function.

For p = 0.05, degree of freedom = 18

Critical value t' = 2.1

Since t < t', the null hypothesis is accepted.

2) For machine utilization:

F-test: H_0: there is no difference between two variances

H_1: variances of two outputs are different

Observed F = $S_1^2/S_2^2 = 0.0258^2/ 0.0211^2 = 1.495$

Critical F (9, 9) at alpha (0.025) = 4.026

Since F < F', the null hypothesis is accepted.

T-test:

H_0: there is no difference between two means

H_1: means of two outputs are different

t = (0.4415 – 0.4406) / SQRT ($0.0258^2/10 + 0.0211^2/10$) = 0.088

For p = 0.05, degree of freedom = 18

Critical value t’ = 2.1

Since t < t’, the null hypothesis is accepted.

Conclusion: For the two parameters compared, there are no significant differences between their means.

6.2.4 Speed comparison based on MicroBlaze soft microprocessor application

As a tentative effort, a soft microprocessor core name MicroBlaze was implemented into FPGA. Also, a simple discrete event simulation model is built using C language, which exactly resembles the one machine model used before (refer to Appendix C). This lean DES model will be used as a control to measure hardware simulator performance. One favorable feature of this DES application is that it can measure the simulation speed more reliably, for the whole microprocessor is dedicated to simulation execution. Results obtained from this model are as follows:

Simulation time: 70000 ns

Clock cycles needed: 70000ns/10ns = 7000

Throughput: 148

Machine utilization: 0.44

Program execution cycles: 4979370

Based on these results, the speedup of hardware simulator over software model is about:

4979370/7000 = 711 (times)

Another interesting experiment is designed as follows: in the DES model, two stochastic variables, inter-arrival time and process time, are substituted by constants. Then the simulation run is timed at 375465 clock cycles. This fact shows that in this specific circumstance, 92% simulation time is spent on random number generation. This surprising result may account for the major reason for the speedup of hardware simulation, and also provide a way of roughly estimating speedup factors. For example, there are 9 stochastic random numbers in two machine hardware simulator vs. 2 stochastic in one machine model, then the speedup should be around 711*9/2 = 3200 times. Based on this observation, it is obvious to predict that reconfigurable hardware simulator of more complex system will result in higher simulation speedup factor.

6.3 Two-Machine manufacturing cell simulation study

6.3.1 Two-Machine cell simulation results

The manufacturing cell model consists of two machine centers, and is capable of simulating up to three different part types. Suppose the arrivals follow exponential distribution, and processing times follow triangular distribution.

Part 1 arrival: exponential (120);

Part 2 arrival: exponential (60);

Part 3 arrival: exponential (90);

Process Plan:

Part 1: 1) Lathe, triangular (15, 20, 25);
        2) Mill, triangular (10, 12, 14);

Part 2: 1) Mill, triangular (17, 20, 23);
        2) Lathe, triangular (10, 15, 20);

Part 3: 1) Mill, triangular (12, 16, 20);
        2) Lathe, triangular (8, 10, 12);
        3) Mill, triangular (12, 16, 20)

Simulation terminating condition: 70000 ns

Replication time: 6958 cycles (69580 ns)

Number of replications: 10

Simulation results are listed in Table 6.4, and statistics are listed in Table 6.5:

	1	2	3	4	5	6	7	8	9	10
P1 in	58	56	64	78	56	49	66	58	50	51
P1 out	57	56	64	78	55	48	65	58	50	48
P2 in	120	118	123	114	133	122	106	102	119	141
P2 out	118	115	119	112	132	122	103	102	117	133
P3 in	82	66	77	83	64	79	89	76	75	87
P3 out	81	65	72	83	64	79	86	74	75	79
L_Busy	3699	3544	3798	4113	3736	3544	3767	3443	3534	3862
M_Busy	5691	5106	5525	5838	5385	5504	5727	5146	5377	5832
L_Util.	0.5316	0.5093	0.5458	0.5911	0.5369	0.5093	0.5414	0.4948	0.5079	0.5550
M_Util.	0.8179	0.7338	0.7941	0.8390	0.7739	0.7910	0.8231	0.7396	0.7728	0.8382

Table 6.4: Two-Machine simulation results

Statistics:

Variable	N	Mean	StDev	SE Mean	95% CI
P1 out	10	57.9	9.21	2.91	(51.3138, 64.4862)
P2 out	10	117.3	10.33	3.27	(109.911, 124.689)
P3 out	10	75.8	7.28	2.30	(70.5888, 81.0112)
L_Util	10	0.5323	0.0285	0.0090	(0.511910, 0.552710)
M_Util	10	0.7923	0.0377	0.0119	(0.765346, 0.819334)

Table 6.5: Two-Machine statistics

6.3.2 Two-Machine cell ARENA model

A Two-Machine manufacturing cell is modeled using ARENA (Figure 6.2)

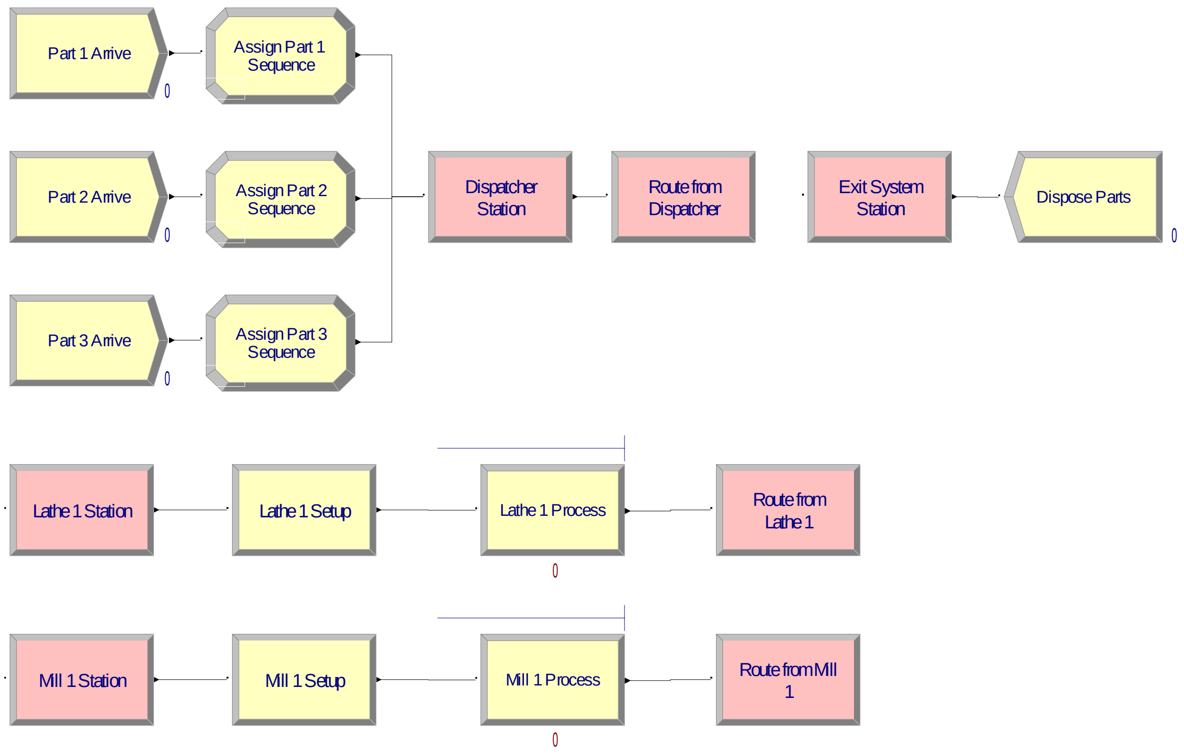


Figure 6.2: Two-Machine-3-Part Arena model

Simulation results and statistics are listed in Table 6.6 and 6.7 respectively:

	1	2	3	4	5	6	7	8	9	10
P1TP	50	50	66	63	68	51	63	67	71	57
P2TP	115	113	114	124	120	128	107	109	114	106
P3TP	76	68	74	78	71	63	71	75	93	70
L_U	0.5042	0.4923	0.5475	0.5628	0.5651	0.5222	0.5232	0.5372	0.5834	0.4921
M_U	0.7691	0.7366	0.7898	0.8317	0.7989	0.7522	0.7480	0.7757	0.8792	0.7304

Table 6.6: Two-Machine cell ARENA model results

P1TP – Part 1 throughput

P2TP – Part 2 throughput

P3TP – Part 3 throughput

L_U – Lathe Utilization

M_U- Mill Utilization

Output Statistics:

Variable	N	Mean	Standard Deviation	SE Mean	95% CI
Part1 Throughput	10	60.6	7.99	2.53	(54.89, 66.31)
Part2 Throughput	10	115.0	7.17	2.27	(109.88, 120.13)
Part3 Throughput	10	73.9	7.98	2.52	(68.19, 79.60)
Lathe Utilization	10	0.5330	0.0317	0.0100	(0.5103, 0.5557)
Mill Utilization	10	0.7812	0.0462	0.0146	(0.7481, 0.8142)

Table 6.7: Two-Machine cell ARENA model statistics

6.3.3 Comparison of Two-Machine cell models

The methodology used to compare two-machine systems is the same as in one-machine case. There are 5 parameters to be compared: part1 throughput, part2 throughput, part3 throughput, lathe utilization and mill utilization. F-test and unpaired T-test are used to judge whether there are differences between their variances and means. Below table list the computation results.

Hypothesis: H_0: there is no difference between two means

H_1: two means are different

	Observed F	Critical F(9,9) Alpha(0.025)	Accept H_0	Observed T	Critical T Alpha(0.05)	Accept H_0
P1_TP	1.33	4.026	Yes	-0.70	2.1	Yes
P2_TP	2.08	4.026	Yes	0.58	2.1	Yes
P3_TP	1.20	4.026	Yes	0.56	2.1	Yes
L_U	1.24	4.026	Yes	-0.005	2.1	Yes
M_U	1.88	4.026	Yes	0.59	2.1	Yes

Table 6.8: Two-Machine cell simulation tests

From the test results listed in Table 6.8, it is concluded that for the five parameters compared, there are no significant differences between their means.

6.3.4 Speed comparison based on MicroBlaze soft microprocessor application

In order to measure software simulation speed, VBA (Visual Basic for Applications) code has been embedded into Arena model. The code will read the system timer at the beginning and ending of the simulation, then calculate simulation time span. All animation and report generation are disabled, and replication is set to 1000. Real time for simulation is measured as 22.5 seconds, which means the speedup is around 5464 times. It appears to be a very good result. However, since there are still other applications running on the PC, and they may interrupt the simulation program every now and then. So the above speedup estimate should be considered optimistic.

6.4 Four-Machine manufacturing system simulation study

In Four-Machine manufacturing system study, the experimental design is similar to that of Two-Machine cell described in last section. There are some minor modifications. In this study, Part 1 arrival follows exponential distribution with mean value of 60 minutes; Part 2 arrival follows exponential distribution with mean value of 90 minutes; and Part 3 arrival follows exponential distribution with mean value of 120 minutes. Part 1 is scheduled to be processed in Cell 1, while Part 2 and Part 3 are scheduled to be processed in Cell 2.

6.4.1 Experiment results

Part 1 arrival: exponential (60);

Part 2 arrival: exponential (90);

Part 3 arrival: exponential (120);

(This sequence is different from the one used in two-machine cell experiments.)

Process Plan:

Part 1: 1) Lathe, triangular (15, 20, 25);

2) Mill, triangular (10, 12, 14);

Part 2: 1) Mill, triangular (17, 20, 23);

2) Lathe, triangular (10, 15, 20);

Part 3: 1) Mill, triangular (12, 16, 20);

2) Lathe, triangular (8, 10, 12);

3) Mill, triangular (12, 16, 20)

Simulation terminating condition: 70000 ns

Replication time: 6958 cycles (69580 ns)

Number of replications: 10

Simulation terminating condition: 70000 ns

Replication time: 6958 cycles (69580 ns)

Number of replications: 10

Simulation results are listed in Table 6.9

	1	2	3	4	5	6	7	8	9	10
P1 out	127	117	118	134	104	95	102	111	110	133
P2 out	65	67	67	76	68	84	74	73	71	85
P3 out	51	61	57	61	57	57	53	50	63	50
L1_Busy	2557	2365	2396	2687	2097	1944	2103	2207	2167	2666
M1_Busy	1450	1404	1434	1625	1256	1141	1238	1330	1310	1597
L2_Busy	1499	1626	1608	1768	1581	1857	1626	1603	1720	1798
M2_Busy	2951	3335	3230	3460	3200	3518	3220	3112	3527	3296
L1_Util	0.3675	0.3399	0.3444	0.3862	0.3014	0.2794	0.3022	0.3172	0.3114	0.3832
M1_Util	0.2084	0.2018	0.2061	0.2335	0.1805	0.1640	0.1779	0.1911	0.1883	0.2295
L2_Util	0.2154	0.2337	0.2311	0.2541	0.2272	0.2669	0.2337	0.2304	0.2472	0.2584
M2_Util	0.4241	0.4793	0.4642	0.4973	0.4599	0.5056	0.4628	0.4473	0.5069	0.4737

Table 6.9: Four-Machine system simulation results

Statistics are listed in Table 6.10:

Variable	N	Mean	StDev	SE Mean	95% CI
Part1 Throughput	10	115.1	13.22	4.18	(105.64, 124.56)
Part2 Throughput	10	73.0	6.99	2.21	(68.00, 78.00)
Part3 Throughput	10	56.0	4.81	1.52	(52.56, 59.44)
Lathe 1 Utilization	10	0.3333	0.0369	0.0117	(0.3069, 0.3597)
Mill 1 Utilization	10	0.1981	0.0222	0.0070	(0.1822, 0.2140)
Lathe 2 Utilization	10	0.2398	0.0161	0.0051	(0.2283, 0.2513)
Mill 2 Utilization	10	0.4721	0.0263	0.0083	(0.4533, 0.4909)

Table 6.10: Four-Machine system simulation statistics

6.4.2 ARENA model and results

6.4.2.1 Building ARENA model

The key part of building this small FMS lies in how to model entity transfer. This function is realized via “leave” and “enter” module. The “leave” module allows one to seize a transferring resource before leaving the station. Here the resource is defined as a robot. Upload time can also be specified. Transfer time will be determined according to robot speed and distance (Table 6.11). The “enter” module allows one to release the transferring resource and provide the option of including an unload delay time, which is defined as zero here. The whole system is illustrated in Figure 6.3

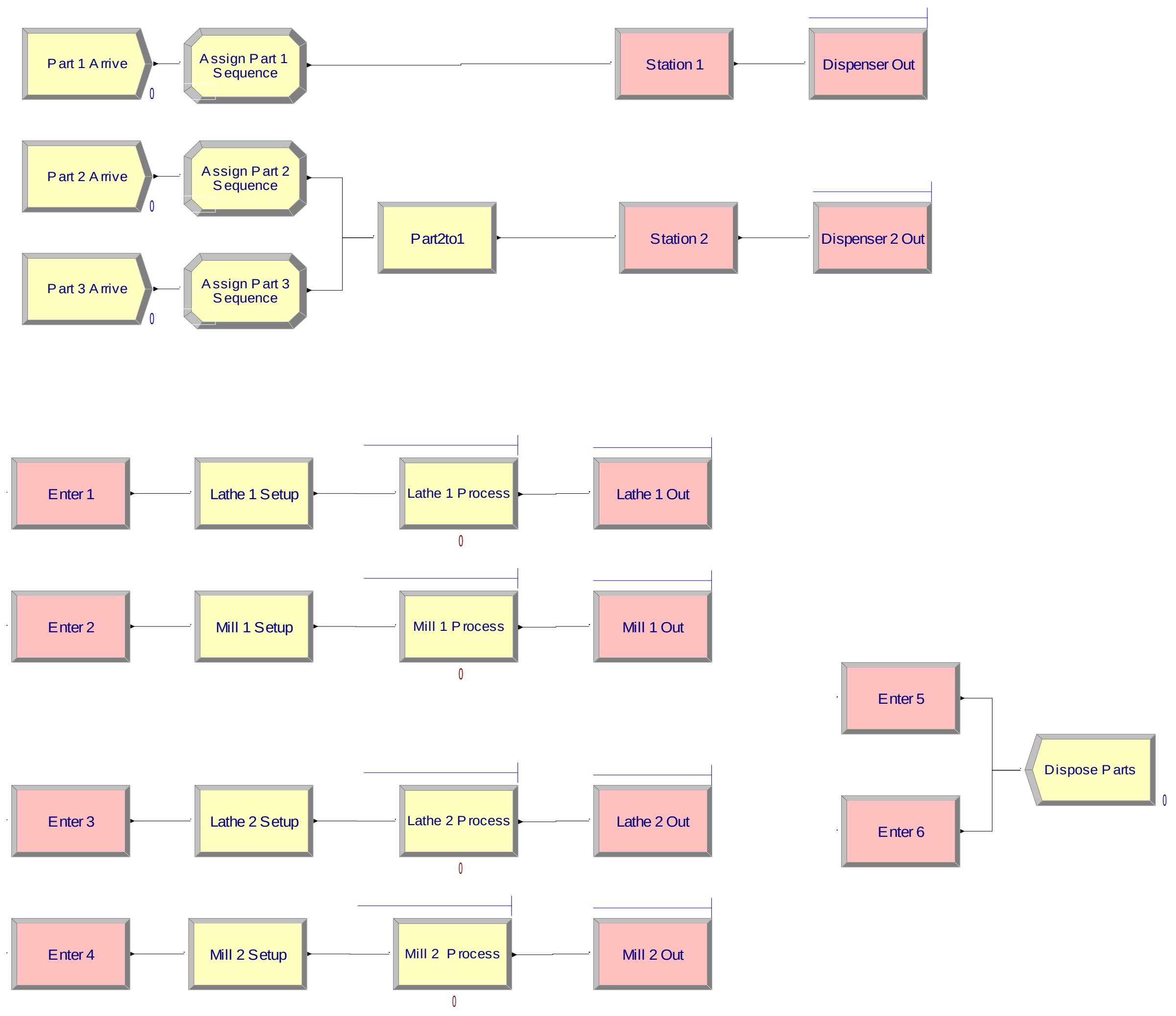


Figure 6.3: Four-Machine FMS ARENA model

	Lathe 1	Mill 1	Lathe 2	Mill 2	Exit
Dispenser	1	2	2	1	3
Lathe 1	NA	1	NA	NA	2
Mill 1	1	NA	NA	NA	1
Lath 2	NA	NA	NA	1	1
Mill 2	NA	NA	1	NA	2

Table 6.11: Machine distance data

6.4.2.2 Simulation results

Simulation results are listed in Table 6.12:

	1	2	3	4	5	6	7	8	9	10
P1TP	132	126	129	103	119	110	117	107	107	109
P2TP	75	79	73	82	73	65	83	75	79	62
P3TP	54	50	61	56	45	59	69	53	52	61
L1_U	0.3876	0.3644	0.3682	0.2934	0.3362	0.3126	0.3313	0.3095	0.3132	0.3153
M1_U	0.2297	0.2188	0.2227	0.1769	0.2044	0.1896	0.2007	0.1860	0.1831	0.1879
L2_U	0.2394	0.2404	0.2490	0.2576	0.2250	0.2239	0.2810	0.2344	0.2495	0.2194
M2_U	0.4546	0.4588	0.4944	0.4913	0.4142	0.4580	0.5637	0.4576	0.4703	0.4620

Table 6.12 Four-Machine ARENA model results

P1TP --- Part 1 throughput

P2TP --- Part 2 throughput

P3TP --- Part 3 throughput

L1_U --- Lathe utilization

M1_U --- Mill utilization

L2_U --- Lathe 2 utilization

M2_U --- Mill 2 utilization

Output Statistics are listed in Table 6.13:

Variable	N	Mean	Standard Deviation	SE Mean	95% CI
Part1 Throughput	10	115.9	10.28	3.25	(108.55, 123.25)
Part2 Throughput	10	74.6	6.83	2.16	(69.71, 79.49)
Part3 Throughput	10	56.0	6.78	2.14	(51.15, 60.85)
Lathe1 Utilization	10	0.3332	0.0307	0.0097	(0.3112, 0.3551)
Mill 1 Utilization	10	0.2000	0.0184	0.0058	(0.1868, 0.2131)
Lathe 2 Utilization	10	0.2420	0.0185	0.0058	(0.2288, 0.2552)
Mill 2 Utilization	10	0.4725	0.0389	0.0123	(0.4447, 0.5003)

Table 6.13: Four-Machine ARENA model statistics

6.4.3 Comparison of Four-Machine system models

There are 5 parameters to be compared: part1 throughput, part2 throughput, part3 throughput, lathe utilization and mill utilization. F-test and unpaired T-test are used to judge whether there are differences between their variances and means. Below table list the computation results.

Hypothesis: H_0: there is no difference between two means

H_1: two means are different

	Observed F	Critical F(9,9) Alpha(0.025)	Accept H_0	Observed T	Critical T Alpha(0.05)	Accept H_0
P1_TP	1.65	4.026	Yes	-0.151	2.1	Yes
P2_TP	1.05	4.026	Yes	-0.517	2.1	Yes
P3_TP	1.99	4.026	Yes	0.0	2.1	Yes
L1_U	1.44	4.026	Yes	0.007	2.1	Yes
M1_U	1.46	4.026	Yes	-0.205	2.1	Yes
L2_U	1.32	4.026	Yes	-0.278	2.1	Yes
M2_U	2.19	4.026	Yes	-0.026	2.1	Yes

Table 6.14: Four-Machine system tests

From the test results listed in Table 6.14, it is concluded that for the seven parameters compared, there are no significant differences between their means. This conclusion indicates that the two models might be identical, thus can be interchangeable in some application areas.

6.4.4 Speedup analysis

In order to measure software simulation speed, VBA (Visual Basic for Applications) code has been embedded into Arena model. The code will read the system timer at the beginning and ending of the simulation, then calculate simulation time span. All animation and report generation are disabled, and replication is set to 1000. Real time for simulation is measured at

35.40 seconds. Since simulation runs on a PC at 1.7G Hz and the FPGA runs at 100M Hz, as a rough estimation:

Speedup = $((35.4/1000)*10^9*17)/70000 = 8597$

However, since there are still other applications running on the PC, and they may interrupt the simulation program every now and then. So the above speedup estimate should be considered optimistic.

6.5 Microprocessor supported physical simulator experiments

After the microprocessor supported simulator design is complete, it is downloaded into the FPGA chip for configuration. Upon the successful configuration, top level simulation procedure starts to run automatically, sending initialization data to hardware simulator. The simulator then carries out high-speed simulation and feedback results to microprocessor. After the simulation ends, microprocessor calculates all required statistics and generates a report. Results from this experiment, including seven parameters, will be compared with those from ARENA model.

6.5.1 Experiments and result

The experiment design is similar to that of logical experiments described in section 6.4.

Part 1 arrival: exponential (60);

Part 2 arrival: exponential (90);

Part 3 arrival: exponential (120);

Process Plan:

Part 1: 1) Lathe, triangular (15, 20, 25);
2) Mill, triangular (10, 12, 14);

Part 2: 1) Mill, triangular (17, 20, 23);
2) Lathe, triangular (10, 15, 20);

Part 3: 1) Mill, triangular (12, 16, 20);
2) Lathe, triangular (8, 10, 12);
3) Mill, triangular (12, 16, 20)

Simulation terminating condition: 70000 ns

Replication time: 6958 cycles (69580 ns)

Number of replications: 10

Simulation terminating condition: 70000 ns

Replication time: 6958 cycles (69580 ns)

Number of replications: 10

Simulation results are shown in Table 6.15:

	1	2	3	4	5	6	7	8	9	10
P1 out	133	103	118	107	104	117	121	112	111	108
P2 out	75	73	71	77	71	79	75	83	90	85
P3 out	53	60	60	43	62	56	45	45	65	41
L1_Util	0.3874	0.2933	0.3359	0.3123	0.2997	0.3435	0.3554	0.3189	0.3212	0.3158
M1_Util	0.2403	0.1771	0.2021	0.1853	0.1805	0.2041	0.2072	0.1894	0.1937	0.1878
L2_Util	0.3643	0.2442	0.2458	0.2288	0.2110	0.2534	0.2376	0.2446	0.2953	0.2437
M2_Util	0.4653	0.4898	0.4895	0.4228	0.4907	0.4902	0.4411	0.4523	0.5670	0.4326

Table 6.15: Hardware simulator output

Output statistics are listed in Table 6.16:

Variable	N	Mean	StDev	SE Mean	95% CI
Part1 Throughput	10	113.4	9.131	2.89	(106.87, 119.93)
Part2 Throughput	10	77.9	6.33	2.00	(73.37, 82.43)
Part3 Throughput	10	53.0	8.84	2.80	(46.67, 59.33)
Lathe 1 Utilization	10	0.3283	0.0281	0.0089	(0.3082, 0.3484)
Mill 1 Utilization	10	0.1967	0.0183	0.0058	(0.1837, 0.2098)
Lathe 2 Utilization	10	0.2569	0.0161	0.0137	(0.2259, 0.2878)
Mill 2 Utilization	10	0.4741	0.0417	0.0132	(0.444305, 0.5040)

Table 6.16: Hardware simulator statistics

6.5.2 Comparison of simulator prototype with ARENA model

The Arena model statistics are shown in Table 6.13. There are 5 parameters to be compared: part1 throughput, part2 throughput, part3 throughput, lathe utilization and mill utilization. F-test and unpaired T-test are used to judge whether there are differences between their variances and means. Below table list the computation results.

Hypothesis: H_0: there is no difference between two means

H_1: two means are different

	Observed F	Critical F(9,9) Alpha(0.025)	Accept H_0	Observed T	Critical T Alpha(0.05)	Accept H_0
P1_TP	1.27	4.026	Yes	-0.575	2.1	Yes
P2_TP	1.18	4.026	Yes	1.12	2.1	Yes
P3_TP	1.70	4.026	Yes	-0.851	2.1	Yes
L1_U	1.19	4.026	Yes	-0.367	2.1	Yes
M1_U	1.01	4.026	Yes	-0.394	2.1	Yes
L2_U	1.26	4.026	Yes	1.00	2.1	Yes
M2_U	1.15	4.026	Yes	-0.091	2.1	Yes

Table 6.17: Hardware simulator tests

From the test results listed in Table 6.17, it is concluded that for the seven parameters compared, there are no significant differences between their means. This conclusion indicates that the software simulation model and hardware simulator can be considered identical in functionality, thus are interchangeable in some application areas.

CHAPTER 7 SUMMARY AND FURTURE RESEARCH DIRECTIONS

7.1 Summary of the new simulation modeling technology

This research presents an innovative simulation modeling technology. A prototype simulation based on this new technology is designed and implemented. Extensive experiments have been done and simulation data are compared with those from software simulation package. Results show that the hardware model and software model are statistically identical, which means they are interchangeable in some application areas. One of the most significant benefits of the hardware modeling technology is speed enhancement. Preliminary result shows that the hardware simulation speedup over traditional software simulation could be several orders of magnitude. In general, hardware simulator can achieve higher speedup for more complex simulation models. This is mainly due to several factors: (1) the reconfigurable device can support massive space parallelism. Simulation events can be processed by many functional units spread out on the PCB board. All event processors are synchronized by one global clock; (2) the simulation processes are mapped into another domain where everything runs as fast as possible, instead of being elaborated by programming language. For example, a part transported from one machine to another in manufacturing system can be mimicked as an electronic signal transmitting from one logic gate to anther on PCB board, whereas, in traditional simulation there have to be steps like reading variable, communicating, message passing, writing back, etc.; (3) since there are no programs running on microprocessor, there will be no time wasted on instruction fetching, decoding and executing, which usually takes up a big portion of execution time of traditional simulation.

The prototype simulator built in this research can run stand-alone without any interaction with microprocessor, thus avoid the inefficiency regarding to software execution. The simulation carried out on this hardware platform is defined as *discrete real-time simulation*, with features like electronic signal representation, binary data transaction, global clock synchronization and scaled real-time execution. The simulator is capable of making trade-offs between accuracy and speed in order to fit for different application areas. Most of the time consuming portions of ordinary PDES, such as event generating and state computing, are realized through hardware connection within one clock cycle. Therefore, tremendous speedup in execution could be achieved. Since global clock is used in simulation, there is no need for complex synchronization algorithms. This feature could avoid significant overhead compared with traditional simulation modeling. There are no event queues in the simulation model, and events are processed immediately when time is due. Therefore, the simulation is deadlock free because there is no waiting needed. The architecture of the simulator can be changed anytime after it is configured, even when the simulator is still running. This feature greatly enhances simulator flexibility, for it is easy to modify the system according to change of environment. The hardware simulator is easy to expand, in other words, has good scalability. However, despite all those advantages, there are also some inherent shortcomings. First, when large time-scaling factor is chosen, the simulation accuracy might be compromised. Second, the hardware simulator utilizes time-stepped mechanism to manage simulation advancement, and this means there could be some idle time between two steps. Therefore, the simulation might not be very efficient while running on single emulation unit. However, in multi-workstation manufacturing system, massive parallelism can overcome this inefficiency. Other drawbacks include resource consuming and not as flexible as software modeling. These can be improved as the reconfigurable computing technology evolves further.

Pros:

- Super fast speed
- Simple synchronization mechanism
- Deadlock free
- Re-configurability
- Massive parallelism
- Scalability
- Binary data representation
- Economically soundness

Cons:

- Insufficient simulation accuracy if large time-scaling factor is chosen
- Inefficient run in single simulation unit

- Resource consuming
- Less flexible than software based simulator

7.2 Significance of this research

This research proposes a new simulation modeling technology especially targeted to time-critical areas where conventional discrete event simulation may not fit. This new simulation modeling technology is significantly different from any traditional discrete event simulator. It employs many innovative concepts and technologies from multidisciplinary areas including industrial engineering, electronic engineering and computer science. To our knowledge, no previous research attempts have been made in this direction. However, as reconfigurable computing technology develops rapidly, hardware cost is going lower and lower and hardware design is becoming much easier. It is believed that this research effort may open a new frontier focused on high performance simulation for time-critical applications.

Although this project is specifically targeted to manufacturing systems, the discrete real-time simulation concept can easily find its applications in many other areas, such as space engineering, automotive industry, land security or military applications. One scenario could be the traffic model of a city during a large event, like a festival parade. Assuming the traffic model is already in place. If a sudden emergency, for example, a terrorist bomb, makes some roads unavailable and requires adding new detours to evacuate people, the new model should be simulated as quickly as possible before implementation. Seconds of delay might cost many lives. Another scenario could be in battle field. If multiple missiles were launched by enemy from various directions toward us, a high-speed simulation is desired to efficiently guide our security system to intercept them as quickly as possible. Such kind of time-critical applications could be easily found in battlefield, space engineering, etc.

7.3 Future research outlook

7.3.1 Adding more components into the simulator

Currently, the target model will be a skeleton image of the desktop FMS. It only consists of machine centers, robots, part dispenser and exit station. To enhance the simulation performance, more details need to be added, such as tool management, pallets, inspection station, etc. All these devices can be modeled via the new technology, and more details can be added to these modules. Once the hardware modeling becomes as easy as software programming, this new technology will become more popular.

7.3.2 Exploring complex routing & scheduling algorithms

Currently only very simple routing and scheduling rules (first-in-first-out) are implemented in the simulator. However, in a real FMS there could be a dozen of different rules available. How to build these rules in low level circuitry remains a tough challenge. Complex heuristics like simulation annealing (SA) [Wrighton and DeHon 2003] and genetic algorithms (GA) [Scott et al. 1995] have been built into FPGA device and several orders of speedup magnitude have been achieved. However, to be utilized in this application, the algorithms must be computed in one clock cycle. A multi-clock system is likely to be feasible.

7.3.3 System-on-a-chip (SOC) design

In future it might be desirable to build a system-on-a-chip (SOC) hardware simulator that can be easily embedded into the FMS system, with Internet accessibility. Nowadays, the leading research and application areas of SOC are the multi-billion dollar telecommunications market. Compared to traditional multi-chip, board-based solutions, the benefits of the SOC include lower cost, faster development, and more flexibility. With the evolution of new technologies in manufacturing area, such as reconfigurable manufacturing and Internet-based simulation, will show more demands on SOC system.

SOC technology is the next step in the evolution of computer science [Lewis, et al, 2002]. Unlike a big chip stuffed mainly with random (glue) logic, SOC is designed as a programmable platform that integrates most of the functions of the end product in a single chip. It incorporates at least one processing element (microprocessor, DSP, etc.) that runs the system’s embedded software. SOC includes peripherals, random logic and interfaces to the outside world and employs a bus-based architecture. It may contain both memory and analog functions. Recently many research efforts have been devoted to SOC technology, and it might provide a new platform for simulation.

REFERENCE

[1] Abumaizar, R. J. and J. A. Svestka, 1997, "Rescheduling job shops under random disruptions", International Journal of Production Research 35 (7), pp2065-2082.

[2] Arnold, J.M, Buell, D.A., Davis, E.G., 1992, "Splash 2", SPAA 4th Annual ACM Symposium on Parallel Algorithms and Architectures, pp316-322.

[3] Askin, R. G., and C. R. Standridge, 1993, "Modeling and Analysis of Manufacturing Systems", John Wiley & Sons, Inc., New York.

[4] Ayani, R., Berkman, B., 1993, "Parallel Discrete Event Simulation on SIMD Computers", Journal of Parallel and Distributed Computing, Volume 18, Number 4, pp501-508.

[5] Balas, E., 1965, "An additive algorithm for solving linear programs with zero-one variables," Operations Research, 13: pp517-546.

[6] Balas, E., 1967, "Discrete programming by the filter method," Operations Research, 15, pp915-957.

[7] Balas, E., 1969, "Machine sequencing via disjunctive graphs: An implicit enumeration algorithm," Operations Research, 17: pp1-10.

[8] Balas, E., 1970, "Machine sequencing: disjunctive graphs and degree-constrained sub-graphs," Naval Research Logistics Quarterly, 17, pp941-957.

[9] Barel, M., 1983, "Fast hardware random number generator for the Tausworthe sequence", Record of Proceedings – 16th Annual Simulation Symposium, pp121-135.

[10] Bauer, J., Bershteyn, M., Kaplan, I., Vyedin, P., 1998, "A reconfigurable logic machine for fast event-driven simulation", In Proceedings of the 1998 35th Design Automation Conference, IEEE, pp668-671.

[11] Beaumont, C., Boronat, P., Champeau, J., Filloque, J.M., Pottier, B., 1994, "Reconfigurable technology: an innovative solution for parallel discrete event simulation support", ACM SIGSIM Simulation Digest, Proceedings of the eighth workshop on Parallel and distributed simulation, Volume 24 Issue 1.

[12] Bhatia, D., 1997, "Reconfigurable computing", VLSI Design, Proceedings of the 10th International Conference, Jan., pp356 – 359.

[13] Bumble, M., 2001, "A parallel architecture for non-deterministic discrete event simulation", PhD dissertation, Computer Science and Engineering, Pennsylvania State University.

[14] Bumble, M., Coraor, L., 1998, “Architecture for a non-deterministic simulation machine”, Winter Simulation Conference Proceedings, v2, pp1599-1606.

[15] Banks, J., Carson, J.S., 1986, “Introduction to discrete-event simulation”, Winter Simulation Conference Proceedings, pp17-23

[16] Chance, F., Robinson, J., and J. Fowler, 1996, “Supporting manufacturing with simulation: model design, development, and deployment”, Proceedings of the 1996 Winter Simulation Conference, pp1-8.

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

[18] Chen, F. F., Xu, D. and Tang, W., 1999, “Enabling Hardware-Based Technologies for Real-Time Simulation and Control of Flexible Manufacturing Systems,” Proceedings of the 1999 International FAIM Conference, pp737-748.

[19] Chin Soon Chong, Appa Iyer Sivakumar, Robert Gay, 2003, “Simulation-based scheduling for dynamic discrete manufacturing”, December, Proceedings of the 35th conference on Winter Simulation, pp1465-1473.

[20] Compton, K., Hauck, S., 2002, “Reconfigurable computing: a survey of systems and software”, ACM Computing Surveys (CSUR), June, Volume 34 Issue 2, pp171-210.

[21] Dado, B., Menhart, P., and Safarik, J., 1993, “Distributed Simulation: A Simulation System for Discrete Event Systems”, Decentralized and Distributed Systems (A-39), pp343-353.

[22] DeHon, A., Wawrzynek, J., 1999, “Reconfigurable computing: what, why, and implications for design automation”, Proceedings of the 36th Design Automation Conference, June, pp610-615.

[23] Drake, G. R., Smith, J. S., Peters, B.A., 1995, “Simulation as a planning and scheduling tool for flexible manufacturing systems”, Proceedings of the 27th Winter Simulation Conference, pp805-812.

[24] Dutta, A., 1990, “Reacting to scheduling exceptions in FMS environments”, IIE Transactions, Vol. 22, No. 4, pp300-314.

[25] Estrin, G., Bussel, B., Turn, R., Bibb, J., 1963, “Parallel processing in a restructurable computer system”, IEEE Transactions on Electronic Computation, pp747-755.

[26] Fujimoto, R. M., 1993, “Parallel and Distributed Discrete Event Simulation: Algorithms and Applications”, Proceedings of the 1993 Winter Simulation Conference, pp106-114.

[27] Fujimoto, R.M., Tsai, J., and Gopalakrishnan, G. 1998, “Design and performance of special purpose hardware for Time Warp”, Proceedings of the 15th Annual Symposium on Computer Architecture, pp 401-408.

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

[29] Jones, A., Rabelo, L., and Y. Yuehwern, 1995, “A Hybrid Approach for Real-Time Sequencing and Scheduling”, International Journal of Computer Integrated Manufacturing, Vol. 8, No 2, pp145-154.

[30] Jones, A., Rabelo, L.C., 1998, "Survey of Job Shop Scheduling Techniques," NISTIR, National Institute of Standards and Technology, Gaithersburg, MD.

[31] Glover, F., 1989, “Tabu search - Part I,” ORSA Journal on Computing, 1 (3): pp190-206.

[32] Glover, F., 1990, “Tabu search - Part II,” ORSA Journal on Computing, 2 (1): pp4-32.

[33] Goldberg, D., 1988, “Genetic Algorithms in Search Optimization and Machine Learning”, Menlo Park: California: Addison-Wesley.

[34] Gonzalez, T. and Sahni, S., 1978, “Flowshop and jobshop schedules: complexity and approximation”, Operations Research, 26, pp36-52.

[35] Grabot, B. and L. Geneste, 1994, “Dispatching rules in scheduling: a fuzzy approach,” International Journal of Production Research, 32 (4): pp903-915.

[36] Greenberg, A.G., Lubachevsky, B.D., Mitrani, I., 1996, “Superfast parallel discrete event simulations”, ACM Transactions on Modeling and Computer Simulation (TOMACS), Volume 6, Issue 2, pp107-136.

[37] Harmonosky, C. M., 1990, "Implemenation issues using simulation for real-time scheduling, control, and monitoring", In Proceedings of the 1990 Winter Simulation Conference, pp595-598.

[38] Harmonosky, C. M., and S. F. Robohn, 1991, “Real time scheduling in computer integrated manufacturing: a review of recent research”, International Journal of Computer Integrated Manufacturing 4: pp331-340.

[39] Harmonosky, C. M., 1995, “Simulation-based real-time scheduling: review of recent developments”, Proceedings of the 27th conference on Winter Simulation, pp220-225.

[40] Harmonosky, C. M., and S. F. Robohn, 1995, “Investigating the application potential of simulation to real-time control decisions”, International Journal of Computer Integrated Manufacturing, vol. 8, pp126-132.

[41] Hauck, S., 1998, “The roles of FPGAs in reprogrammable systems”, Proceedings of the IEEE, Volume: 86, Issue: 4, pp615 – 638.

[42] Heidelberger, P., 1986, “Statistical Analysis of Parallel Simulations”, Proceedings of 1986 Winter Simulation Conference, pp290-295.

[43] Jain, A. K., and Elmaraghy, H. A., 1997, “Production scheduling/rescheduling in flexible manufacturing”, International Journal of Production Research, Vol. 35, No. 1, pp281-309.

[44] Jefferson, D. R., 1985, “Virtual Time, ACM Transactions on Programming Language and Systems”, Vol. 7, No. 3, pp404-425

[45] Jefferson, D., Beckman, B., Wieland, F., Blume, L., DiLoreto, M., Hontalas, P., Laroche, P., Sturdevant, K., Tupman, J., Warren, V., Wedel, J., Younger, H., and Bellenot, S., 1987, “Distributed Simulation and the Time Warp Operating System,” Proceedings of the 12th SIGOPS — Symposium on Operating Systems Principles, pp. 77-93.

[46] Kelton, W. D., 2002, “Simulation with Arena”, Second Edition, McGraw Hill, ISBN 0-07-239270-3.

[47] Kim, M. H., and Y. Kim, 1994, “Simulation based real time scheduling in a flexible manufacturing system”, Journal of Manufacturing Systems 13 (2): pp85-93.

[48] Kirkpatrick, S., C. Gelatt and M. Vecchi, 1983, “Optimization by simulated annealing,” Science, 220 (4598): pp671-680.

[49] Konas, P., Yew, P.C, 1991, “Parallel discrete event simulation on shared-memory multiprocessors”, ACM SIGSIM Simulation Digest archive, Volume 21, Issue 3, pp134 – 148.

[50] Koren, Y. and Ulsoy, A.G., 1997, “Reconfigurable Manufacturing Systems”, Engineering Research Center for Reconfigurable Machining Systems (ERC/RMS) Report #1, The University of Michigan, Ann Arbor, MI.

[51] Kumar, D., 1986, “Simulating feed-forward systems using a network of processors”, Proceedings of the 19th annual symposium on Simulation, pp127 – 144.

[52] Law, A.M., Michael G. McComas, 1999, “Simulation of manufacturing systems”, Proceedings of the 31st conference on Winter Simulation, pp49-52.

[53] Lewis, B., Bolsens, I., Lauwereins, R., Wheddon, C., Gupta, B., Tanurhan, K., 2002, "Reconfigurable SoC - What Will it Look Like?", March, Proceedings of the conference on Design, automation and test in Europe, pp660-663.

[54] Lewis, T.G. and Payne, W.H., 1973, "Generalized feedback shift register pseudorandom number algorithm", Journal of the Association for Computing Machinery, 20(3): pp456-468.

[55] Li, R.K., Shyu, Y.T., and Adiga, S., 1993, "A heuristic rescheduling algorithm for computer-based production scheduling systems," International Journal of Production Research, Vol. 31, 1815-1826.

[56] Lyons, R.G., 1996, "Understanding Digital Signal Processing", Addison Wesley.

[57] Matsuura, H., Tsubone,H. and Kanezashi, M., 1993, "Sequencing, dispatching, and switching in a dynamic manufacturing environment," International Journal of Production Research, Vol. 31, pp1671-1688.

[58] McCollum, J.M., Lancaster, J.M., Peterson, J.M, Gregory, D., 2003, "Using Reconfigurable Computing to Accelerate Simulation Applications", Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms, pp308-311.

[59] Mehrabi, M.G., Ulsoy, A.G., Y.Koren and P.Heytler, 2002, "Trends and perspectives in flexible and reconfigurable manufacturing systems", Journal of Intelligent Manufacturing, 13, pp135-146.

[60] Miller, S., Pegden, D., 2000, "Introduction to manufacturing simulation," Proceedings of the 32nd conference on Winter Simulation, pp63-66.

[61] Misra, J., 1986, "Distributed Discrete-Event Simulation," ACM Computing Surveys, Vol. 18, No. 1, pp. 39-65.

[62] Montazer, M., Wassenhove, L.V., 1990, "Analysis of Scheduling rules for an FMS", International Journal of Production Research", 28, pp785-802.

[63] Page, E. H., Nance, R. E., 1994, "Parallel Discrete Event Simulation: A Modeling Methodological Perspective", Proceedings of 8th workshop parallel distributed simulation, pp88-93.

[64] Page, E.H., 1999, "Beyond speedup: PADS, the HLA and Web-based simulation", Proceedings of the thirteenth workshop on Parallel and distributed simulation, pp2-9.

[65] Rabelo, L., 1990, “Hybrid Artificial Neural Networks and Knowledge-Based Expert Systems Approach to Flexible Manufacturing System Scheduling,” PhD. Dissertation, University of Missouri-Rolla.

[66] Reed, D.A., Malony, A.D., McCredie, B.D., 1987, “Parallel discrete event simulation: a shared memory approach”, ACM SIGMETRICS Performance Evaluation Review, Proceedings of the 1987 ACM SIGMETRICS conference on Measurement and modeling of computer systems, Volume 15 Issue 1, pp36-38.

[67] Reynolds, P.F. 1988, “A spectrum of options for parallel simulation”, Proceedings of the 20th conference on Winter Simulation, pp325-pp332.

[68] Reynolds, P.F., Pancerella, C.M., Srinivasan, S., 1992, “Making parallel simulations go fast”, Proceedings of the 24th conference on Winter Simulation, pp646-pp656.

[69] Rogers, P., Gordon, R. J., 1993, “Simulation for real-time decision making in manufacturing systems”, Proceedings of the 25th conference on Winter Simulation, pp866-874.

[70] Rose, J., Gamal, A.E., and Sangiovanni-Vincentelli, A., 1993, “Architecture of field programmable gate arrays,” Proc. IEEE, vol. 81, pp1013–1029.

[71] Sabuncuoglu, I., and M. Bayiz, 2000, “Analysis of reactive scheduling problems in a job shop environment”, European Journal of Operations Research 126 (3), November, pp567-586.

[72] Scott, S.D., Samal, A., Seth, S., 1995, “HGA: a hardware-based genetic algorithm”, February, Proceedings of the 1995 ACM third international symposium on Field-programmable gate arrays, pp53-59.

[73] Smith, J. S., Wysk, R. A., Sturrock, D. T., Ramaswamy, S. E., Smith, G. D., and S.B. Joshi, 1994, “Discrete Event Simulation for Shop Floor Control”, In Proceedings of the 1994 Winter Simulation Conference, pp962-959.

[74] Srinivasan, V., 1971, “A hybrid algorithm for the one machine sequencing problem to minimize total tardiness,” Naval Research Logistics Quarterly, 18: pp317-327.

[75] Tabe, T., and Salvendy, G., 1988, “Toward a hybrid intelligent system for scheduling and rescheduling of FMS”, International Journal of Computer Integrated Manufacturing, Vol. 1, No. 3, pp154-164.

[76] Tausworthe, R.C., 1965, “Random numbers generated by linear recurrence modulo two”, Mathematics of Computation, 19: pp201-209.

[77] Teo, Y.M., Y. K. Ng, Y.K., Onggo, B.S.S., 2002, “Conservative simulation: Conservative simulation using distributed-shared memory”, Proceedings of the sixteenth workshop on Parallel and distributed simulation, pp3-10.

[78] Thompson, M. B, 1994, “Expanding Simulation Beyond Planning and Desing”, Industrial Engineering, Vol. 26, No. 10, pp65-67.

[79] Tomita, M., Suganuma, N., Hirano, K., 1993, “Reconfigurable machine and its application to logic simulation”, IEICE Transactions on Fundamentals of Electronics Communications and Computer Science, pp1705-1712.

[80] Tsujimura, Y., S. Park, S. Chang and M. Gen, 1993, “An effective method for solving flow shop scheduling problems with fuzzy processing times,” Computers and Industrial Engineering, 25: pp239-242.

[81] Vakili, P., 1992, “Massively parallel and distributed simulation of a class of discrete event systems: a different perspective”, ACM Transactions on Modeling and Computer Simulation (TOMACS), Volume 2 Issue 3, pp214-238.

[82] Vieira, G. E., Herrmann, J. W., and Edward Lin, 2003, “Rescheduling manufacturing systems: a framework of strategies, policies, and methods”, Journal of Scheduling, Vol. 6, pp39-62.

[83] Wang, J. J., Abrams, M., 1992, “Approximate Time-Parallel Simulation of Queuing Systems with Losses”, Proceedings of the 1992 Winter Simulation Conference, pp700-708.

[84] Wrighton, M.G., Dehon, A.M., 2003, “Hardware-assisted simulated annealing with application for fast FPGA placement”, February, Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field programmable gate arrays, pp33-42.

[85] Wu, D. 1987, “An Expert Systems Approach for the Control and Scheduling of Flexible Manufacturing Systems, Ph. D Dissertation, Pennsylvania State University.

[86] Xilinx, 2003, “MicroBlaze Processor Reference Guide”, www.xilinx.com/ise/embedded/edk6_2docs/mb_ref_guide.pdf.

[87] Xilinx, 2004 (1), “Connecting Customized IP to the MicroBlaze Soft Processor Using the Fast Simplex Link (FSL) Channel”, XAPP529 (v1.3), May, http://direct.xilinx.com/bvdocs/appnotes/xapp529.pdf.

[88] Xilinx, 2004 (2), “Virtex-4 Family Overview”, DS112 (v1.2), December, http://direct.xilinx.com/bvdocs/publications/ds112.pdf.

[89] Xilinx, 2005 (1), “Custom Peripheral Design Guide”, http://direct.xilinx.com/direct/ise7_tutorials/import_peripheral_tutorial.pdf.

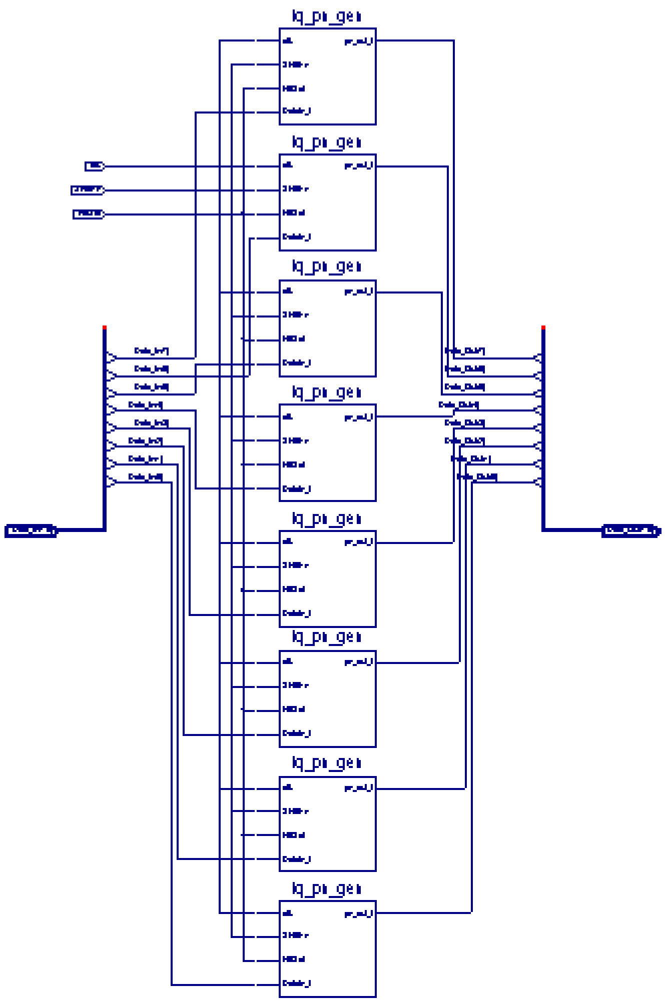
[90] Xilinx, 2005 (2), “Platform Studio User Guide”, http://www.xilinx.com/ise/embedded/ps_ug.pdf.

[91] Xilinx, 2005 (3), “Virtex-II Platform FPGAs: Complete Data Sheet”, http://direct.xilinx.com/bvdocs/publications/ds031.pdf.

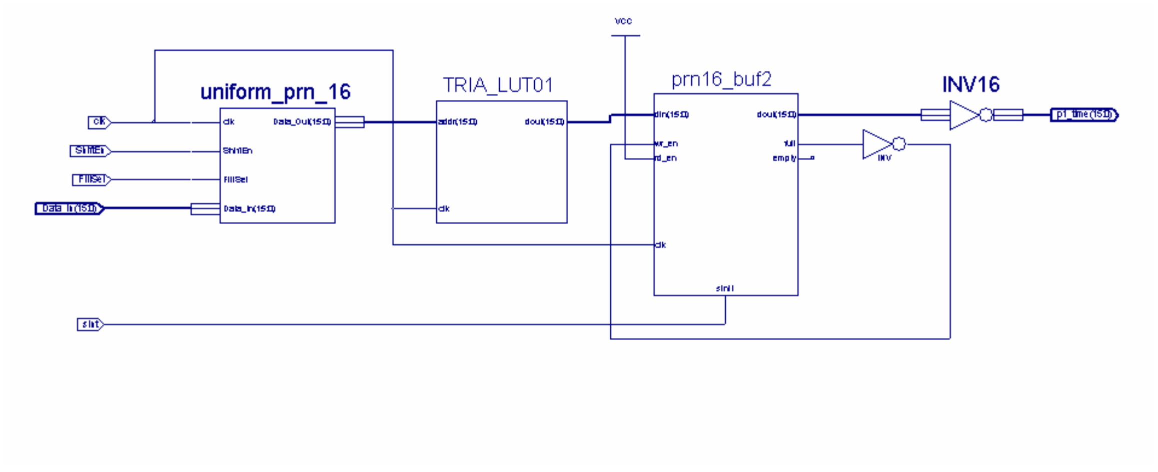
[92] Xu, D., 2001, “Hardware-based Parallel Simulation of Flexible Manufacturing Systems,” PhD Dissertation, Industrial & Systems Engineering Department, Virginia Tech.

[93] Yamamoto, M. and Nof, S. Y., 1985, “Scheduling and rescheduling in the Manufacturing operating system environment”, International Journal of Production Research, Vol. 23, pp705-722.

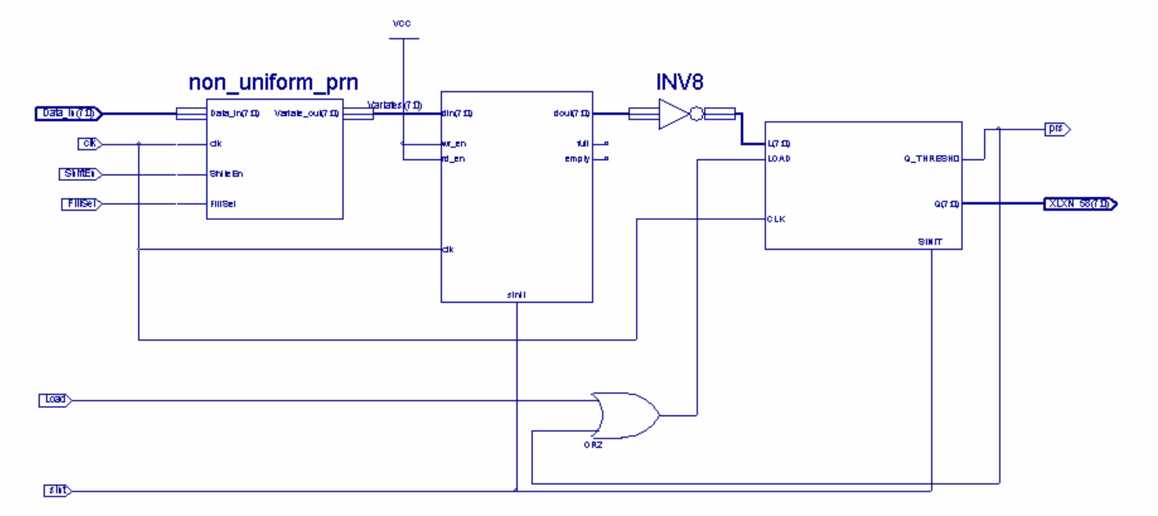
APPENDIX A: Design Schematics



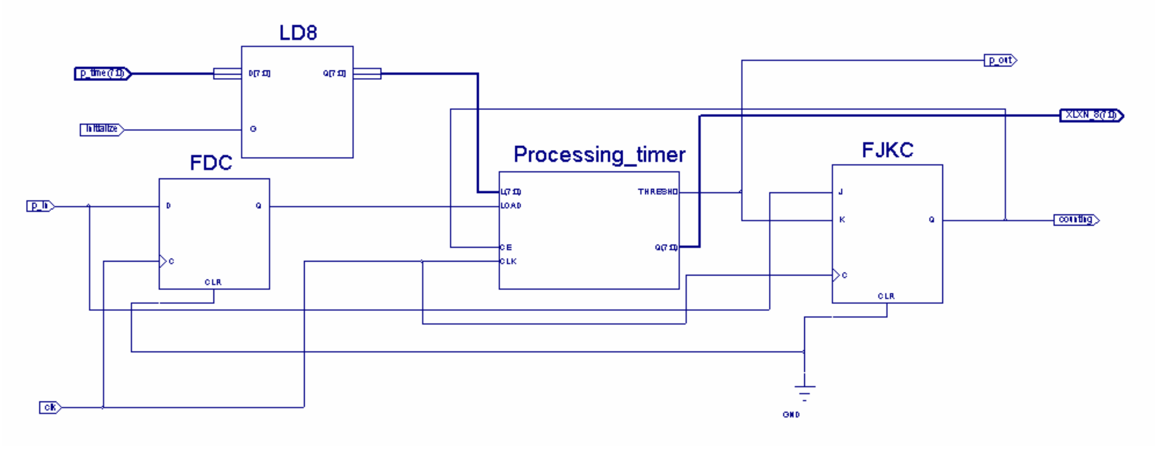
Multi-Bit Random Number Generator



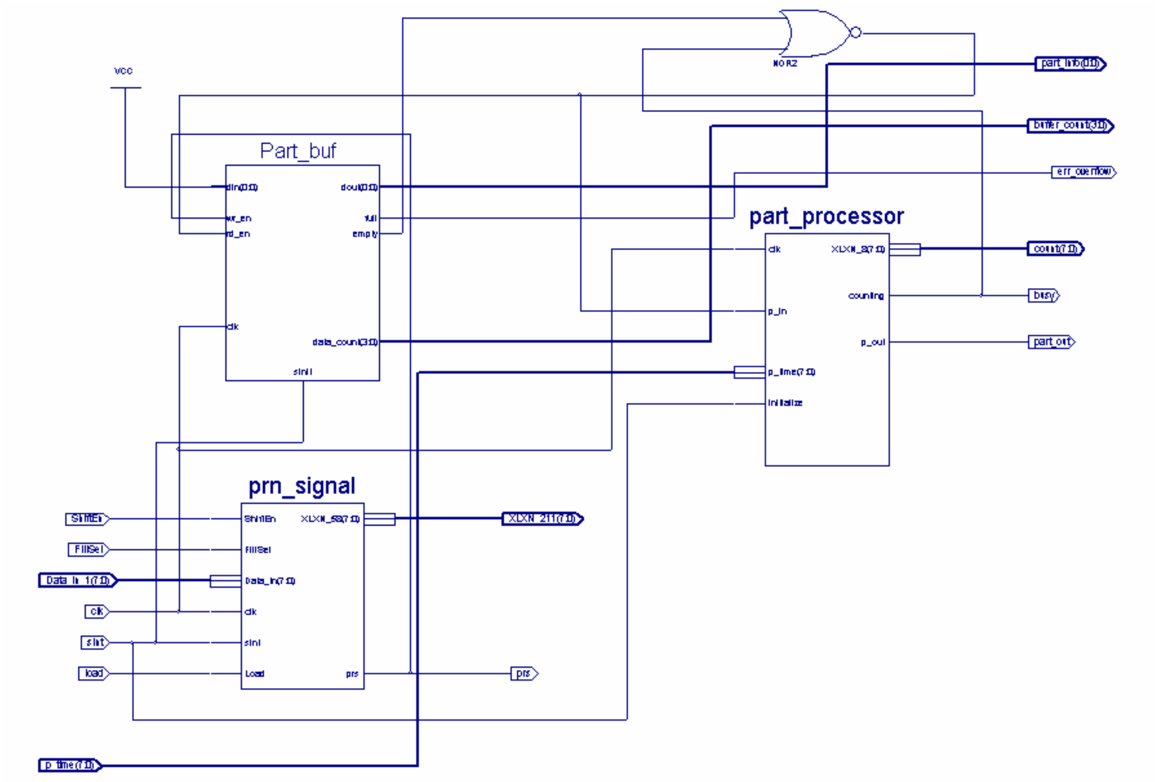
Non-uniform Random Variate Generator



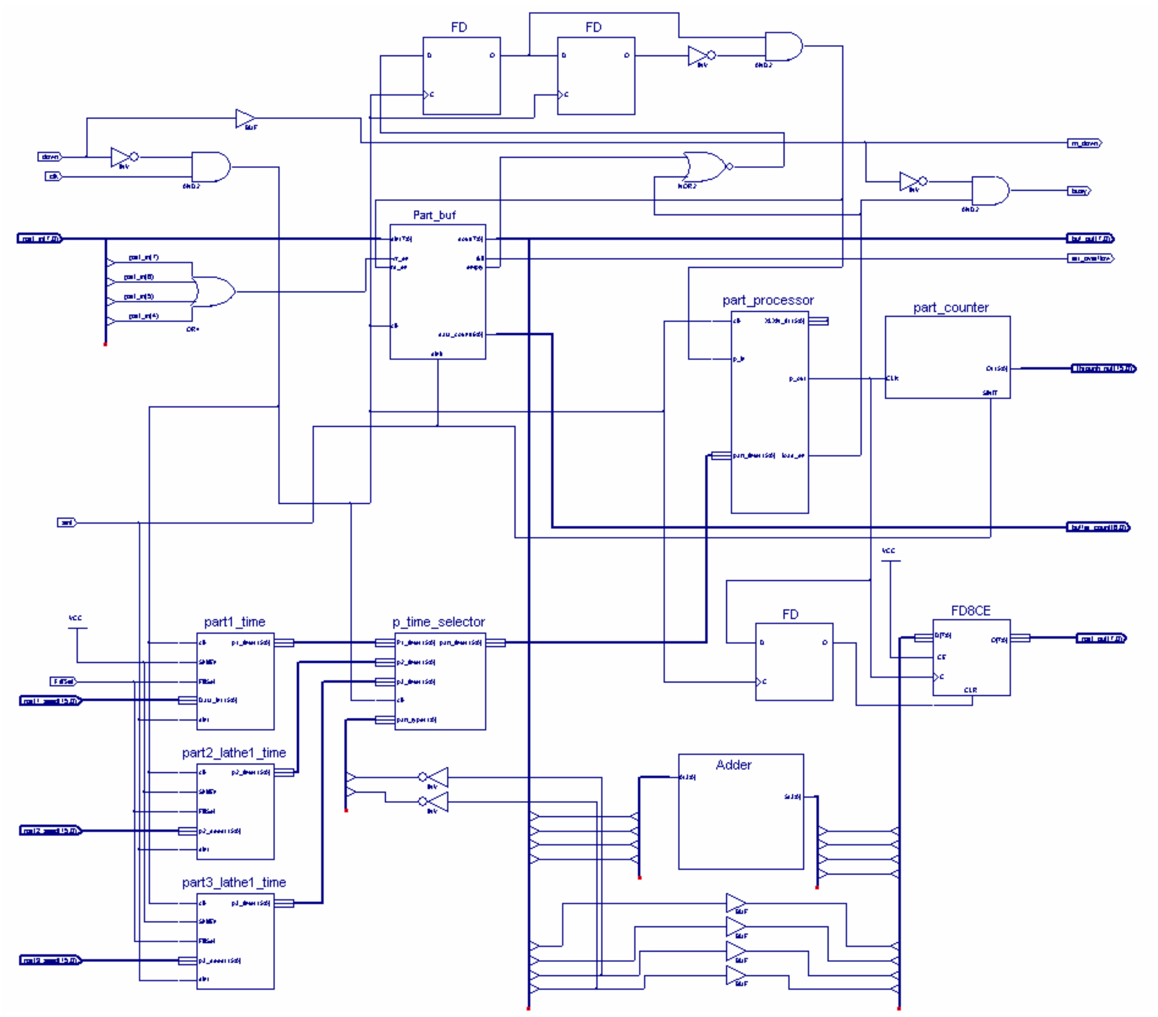
Part Signal Generator



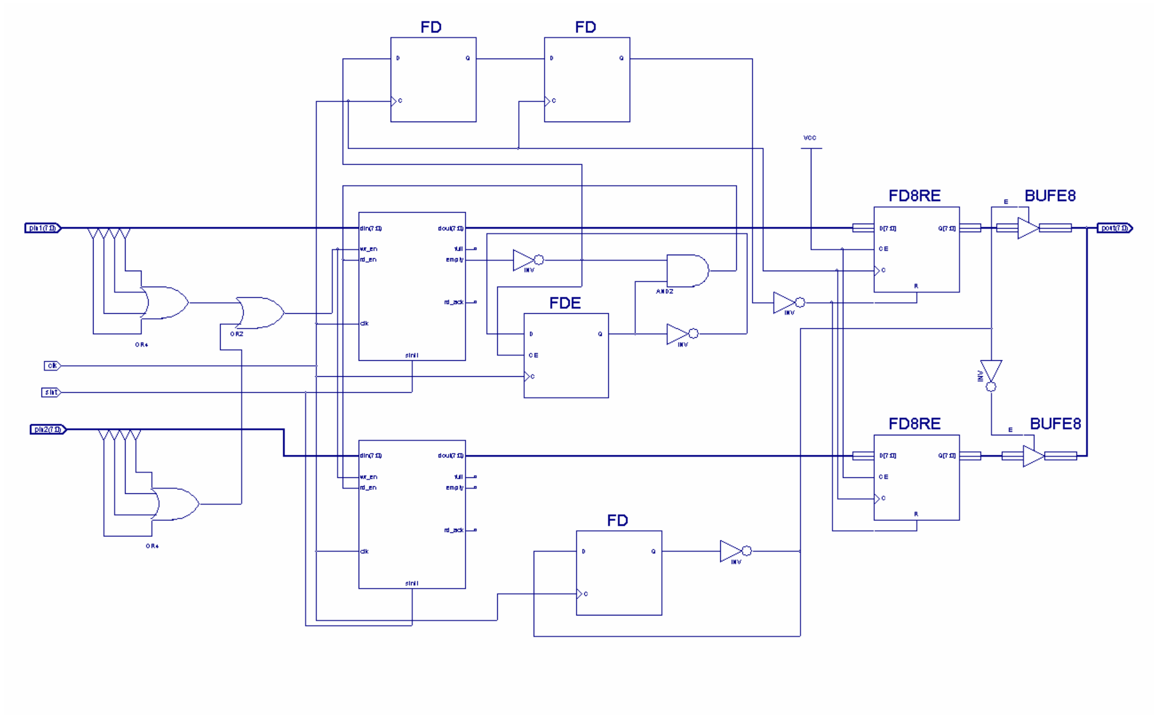
Part Processor



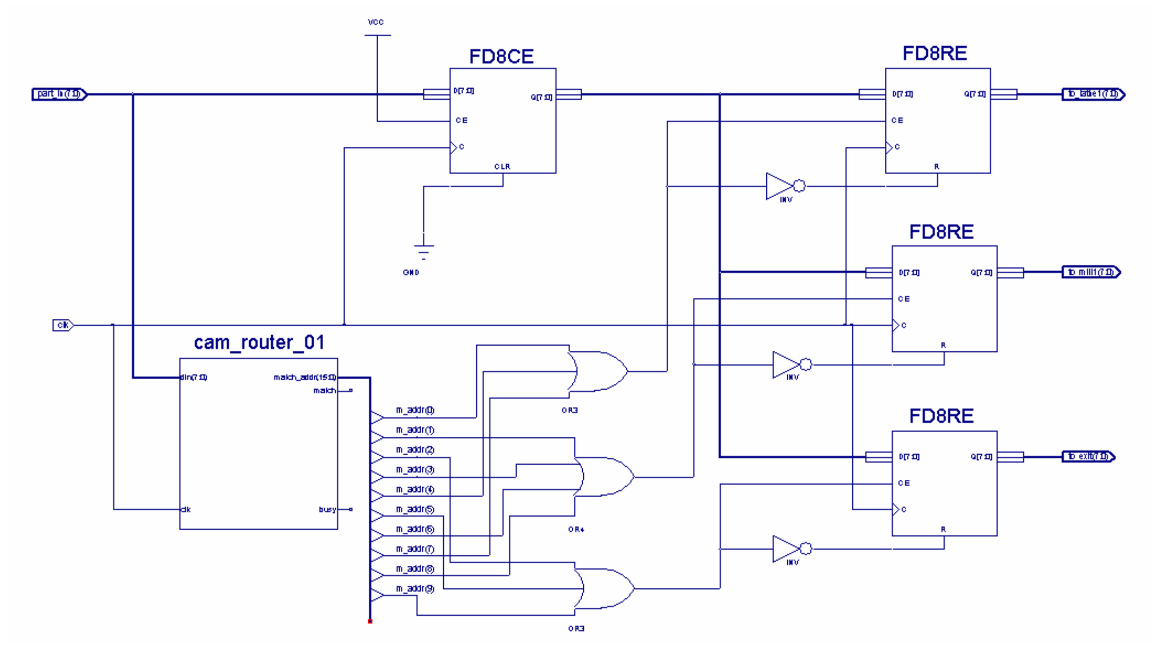
One-Machine-Signal-Part model



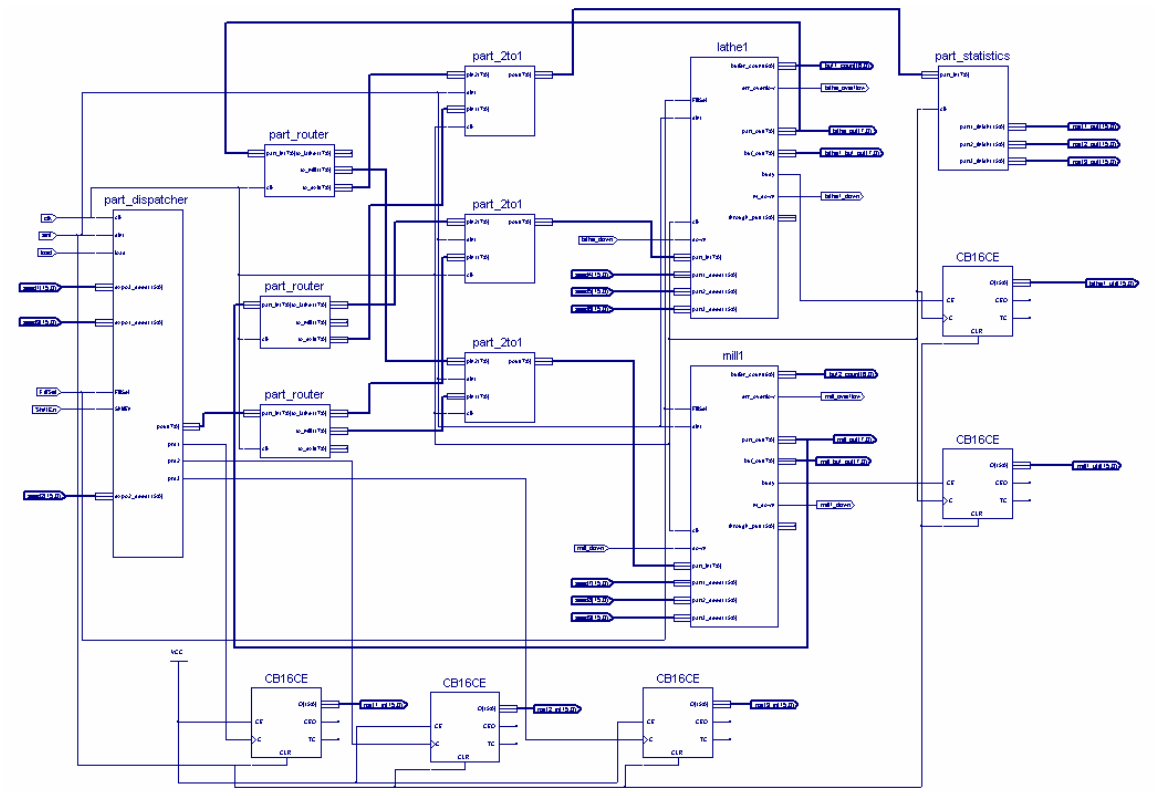
One-Machine-Three-Part model



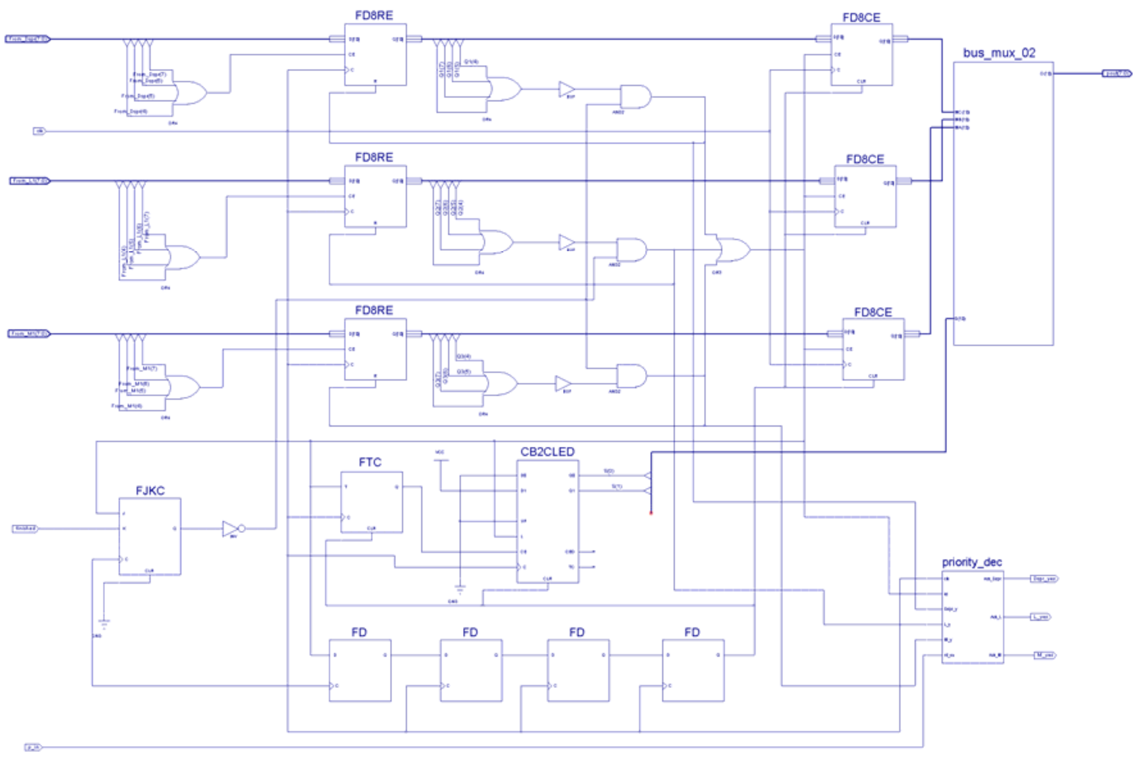
Part_2to1



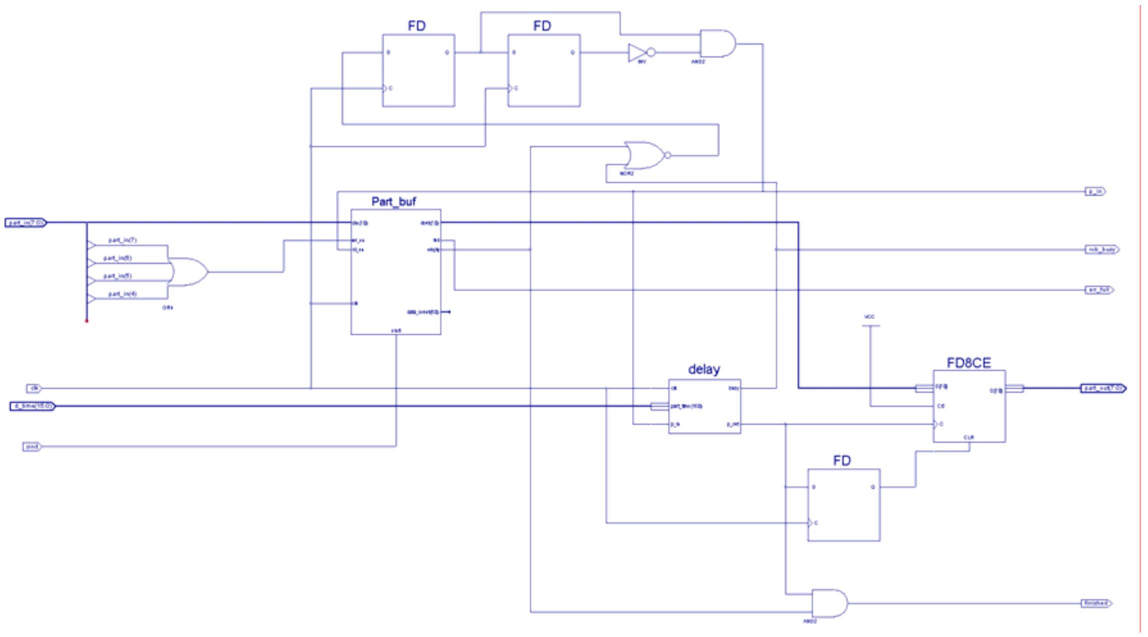
Part Router



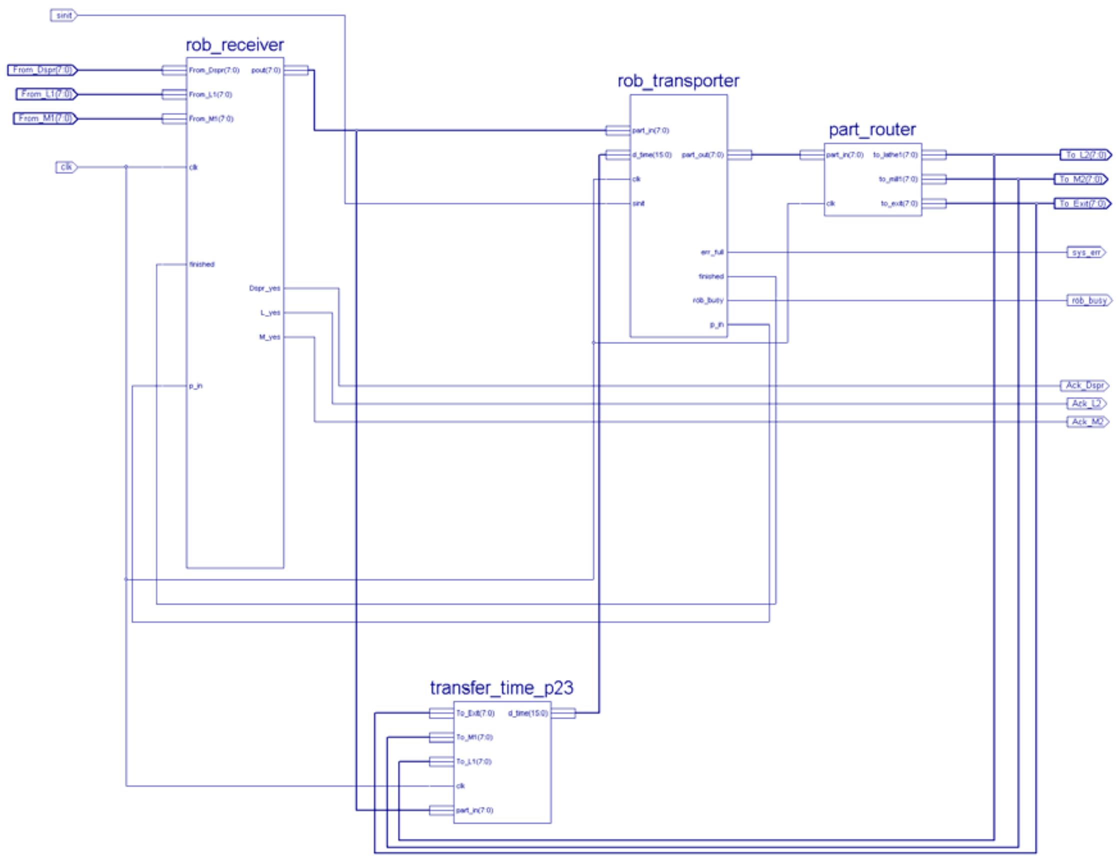
Two-Machine Manufacturing Cell model



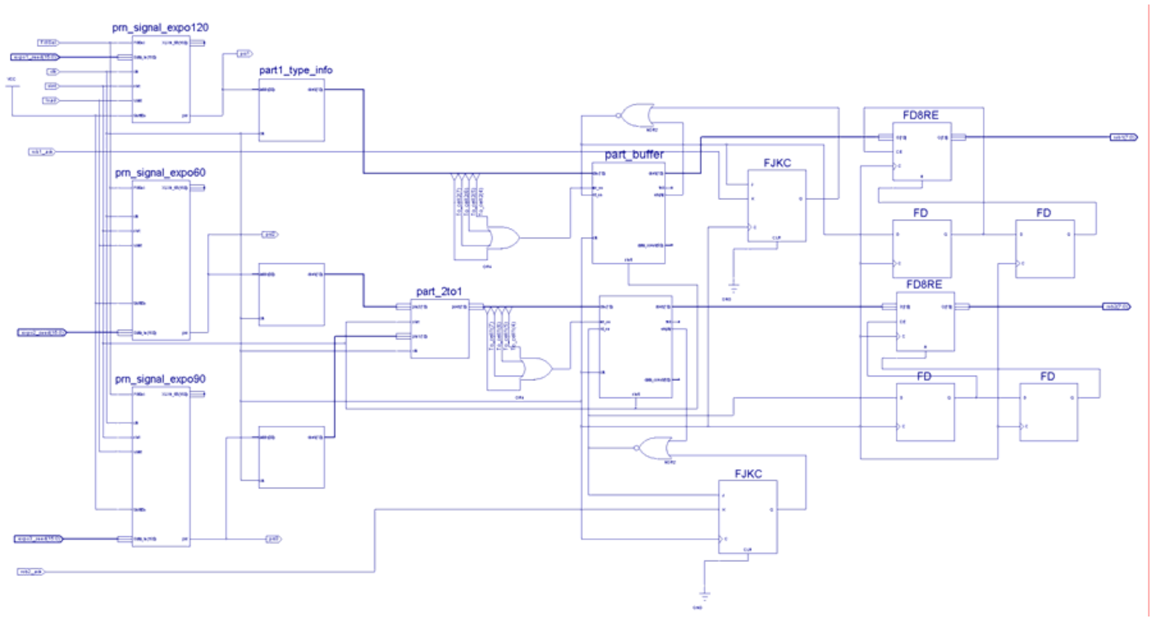
Robot Receiver



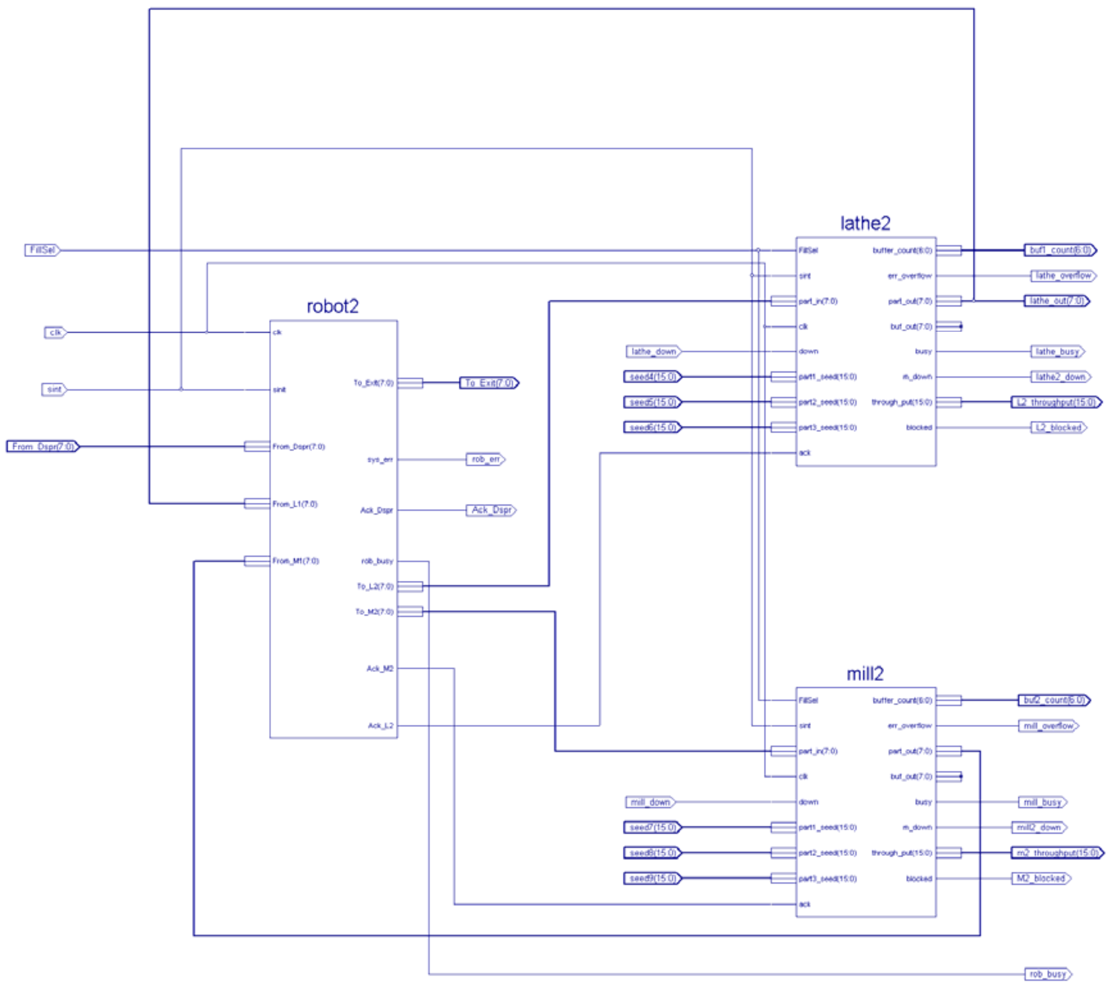
Robot Transporter


sinit
rob_receiver
From_Dspr(7:0)
From_L1(7:0)
From_M1(7:0)
clk
From_Dspr(7:0)
pout(7:0)
From_L1(7:0)
From_M1(7:0)
clk
finished
Dspr_yes
L_yes
M_yes
p_in
rob_transporter
part_in(7:0)
d_time(15:0)
part_out(7:0)
clk
sinit
err_full
finished
rob_busy
p_in
part_router
part_in(7:0)
to_lathe1(7:0)
to_mill1(7:0)
clk
to_exit(7:0)
To_L2(7:0)
To_M2(7:0)
To_Exit(7:0)
sys_err
rob_busy
Ack_Dspr
Ack_L2
Ack_M2
transfer_time_p23
To_Exit(7:0)
d_time(15:0)
To_M1(7:0)
To_L1(7:0)
clk
part_in(7:0)

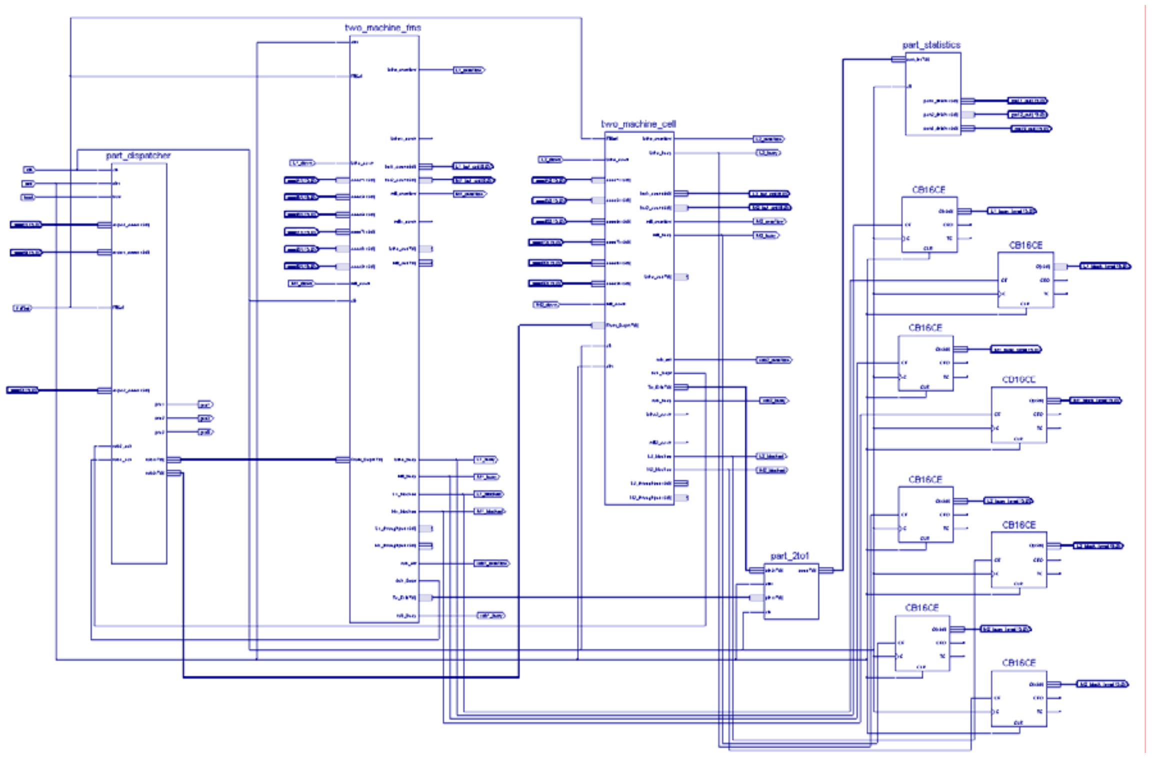
Robot model



Part Dispatcher

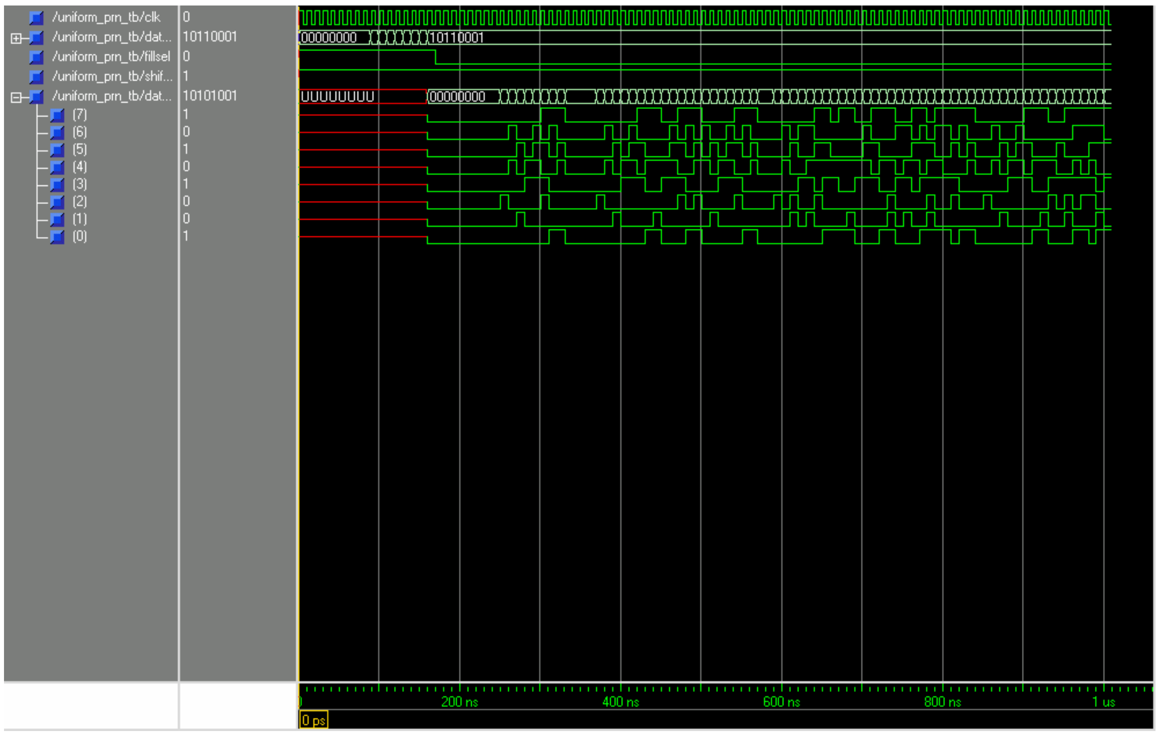


Two-Machine Cell model (with robot)

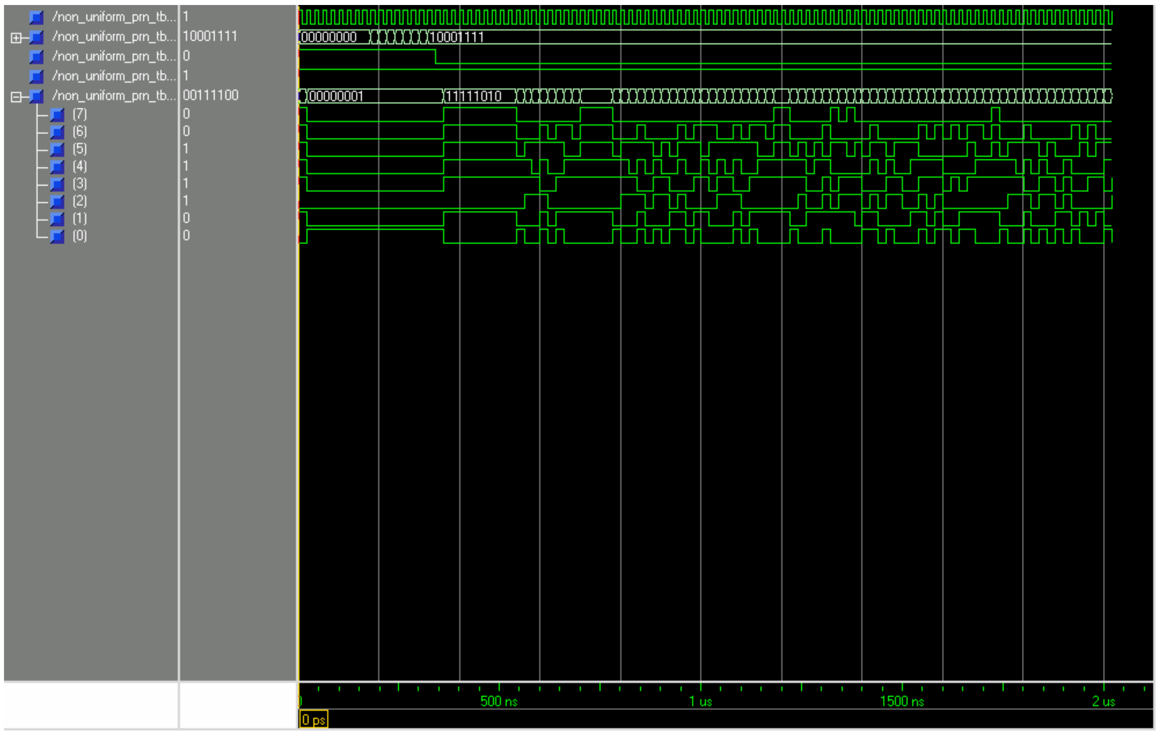


Four-Machine FMS model

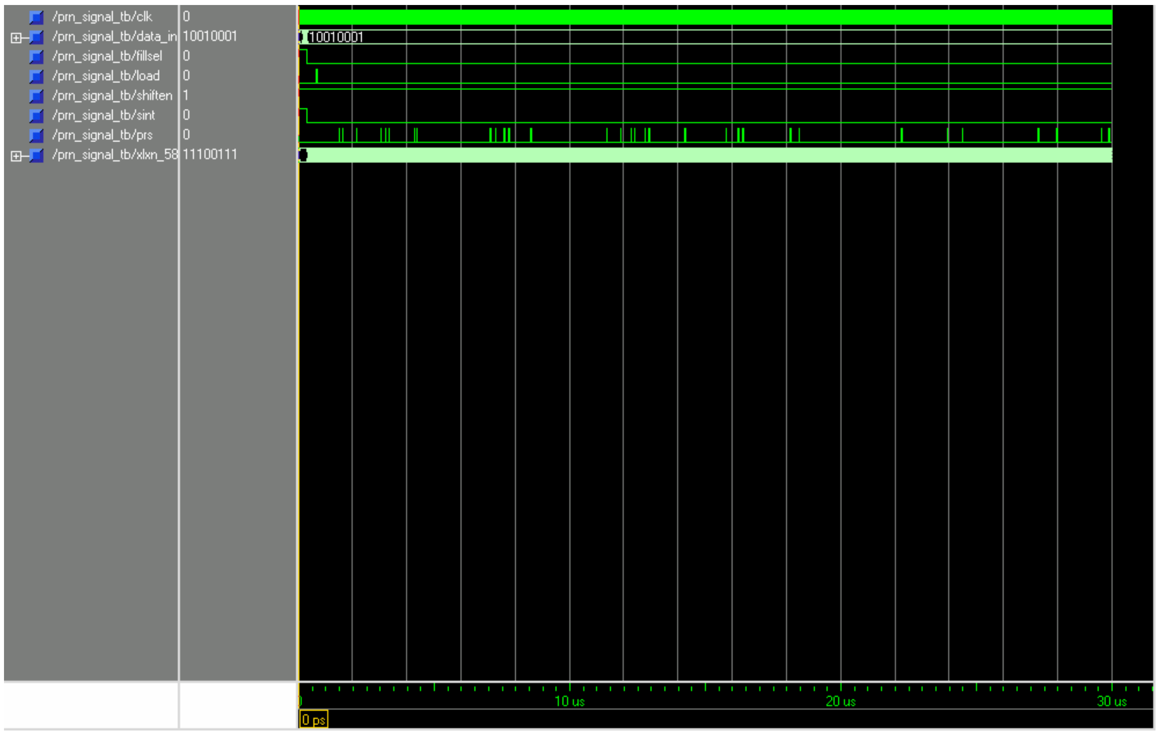
APPENDIX B: Output Waveforms



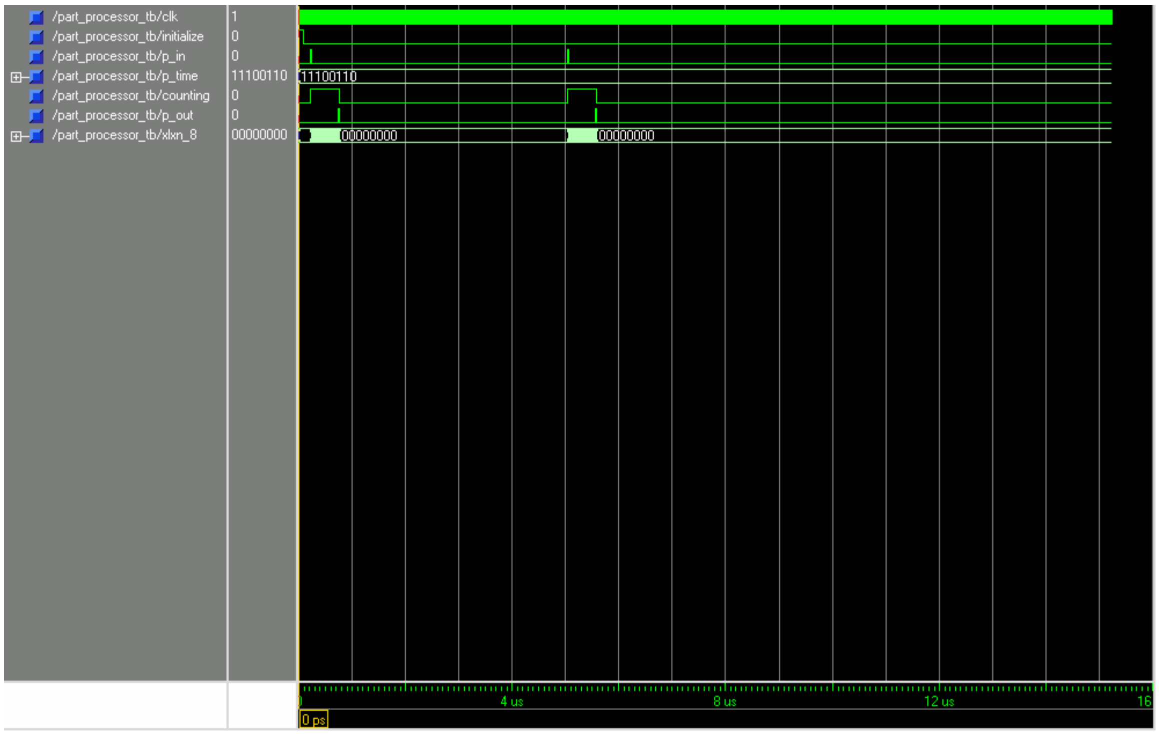
Uniform random number generator



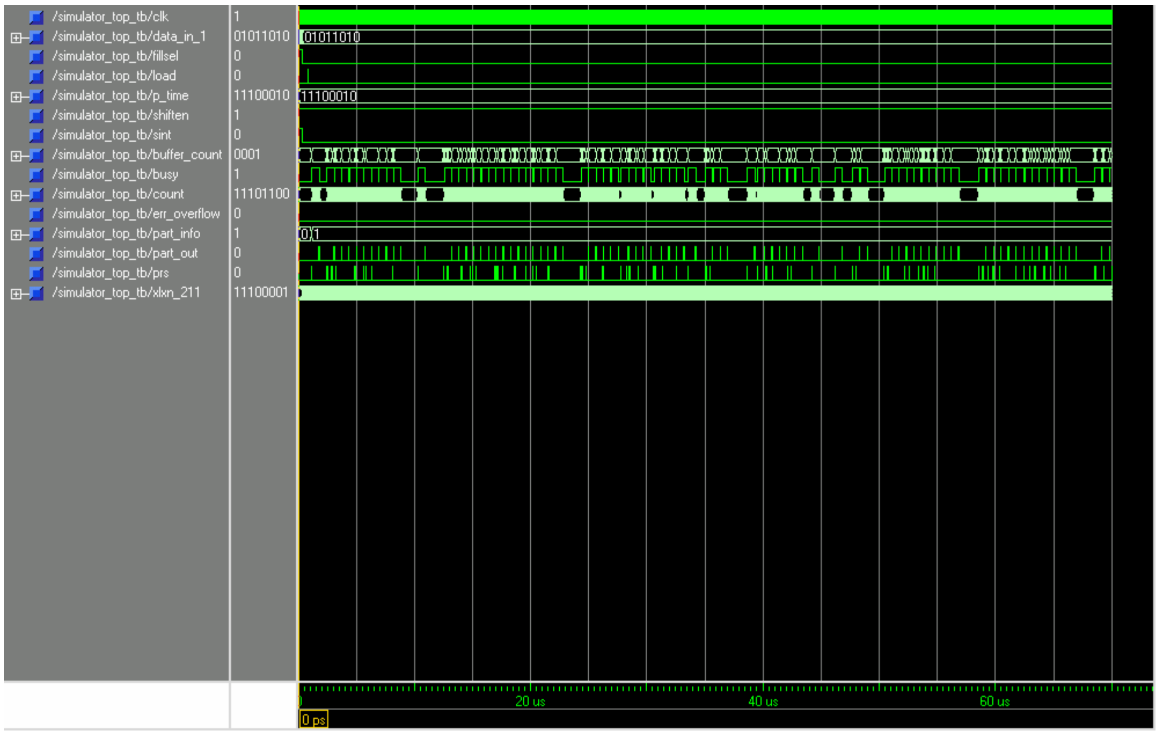
Non-uniform random numbers (exponentially distributed variates)



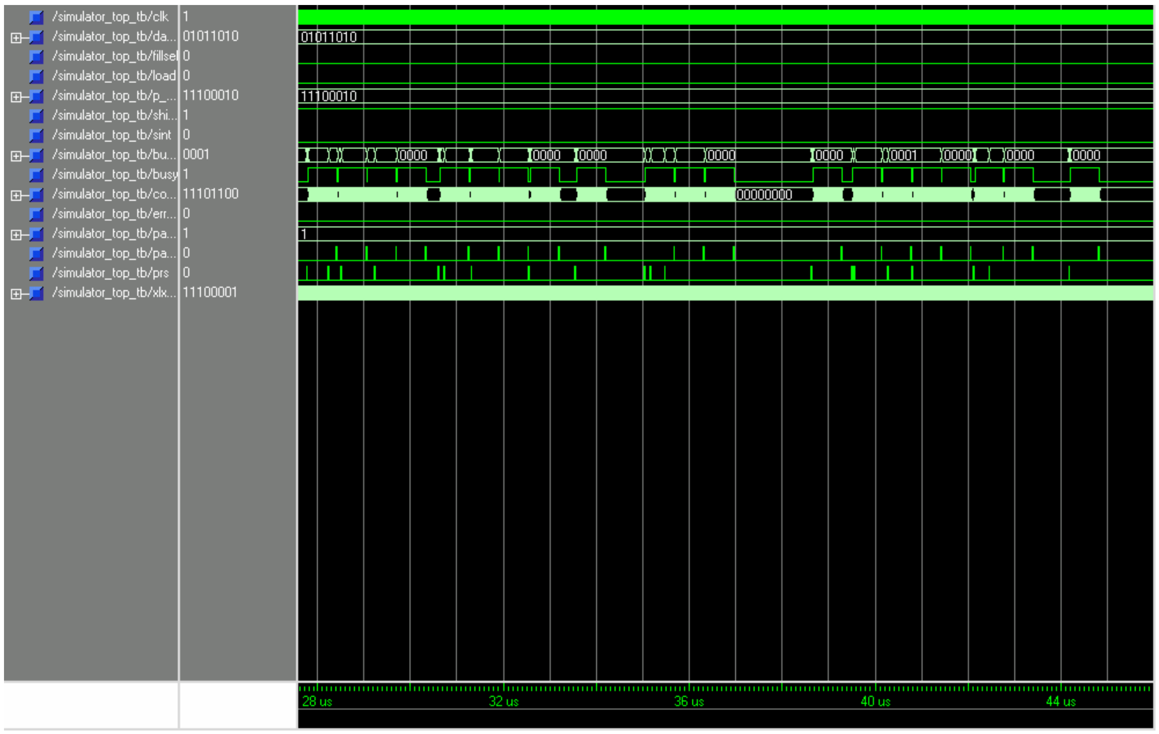
Part signal waveform



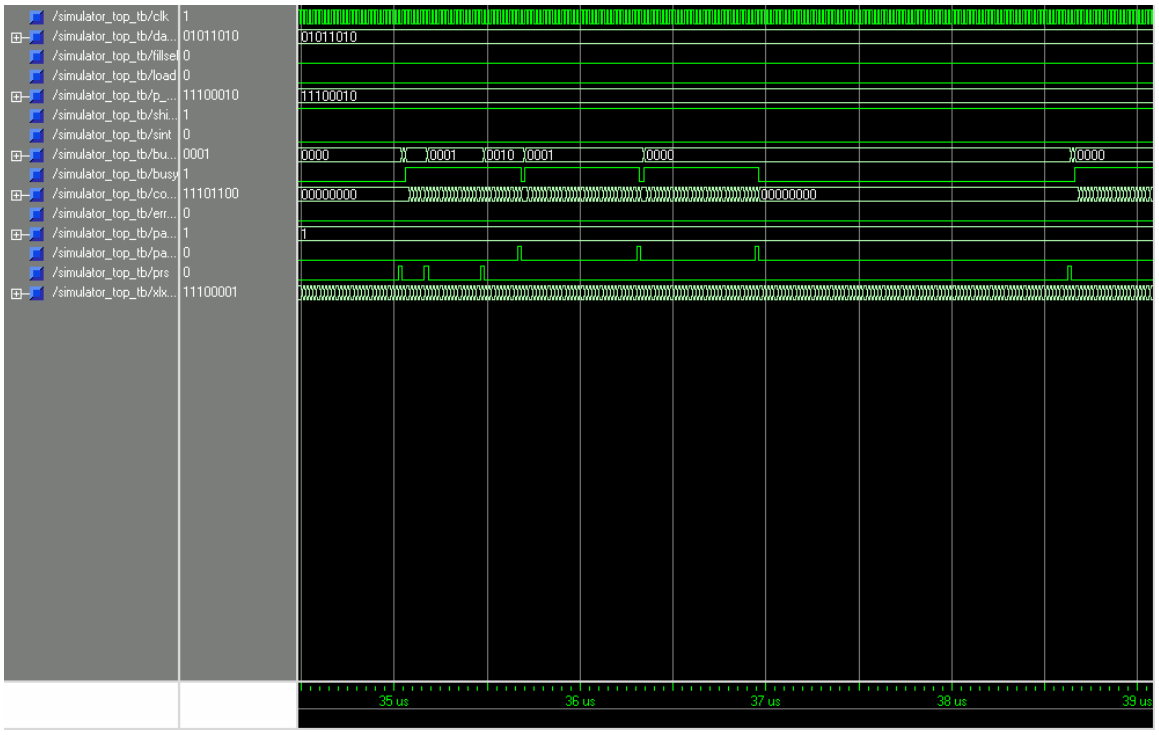
Part Processor waveform



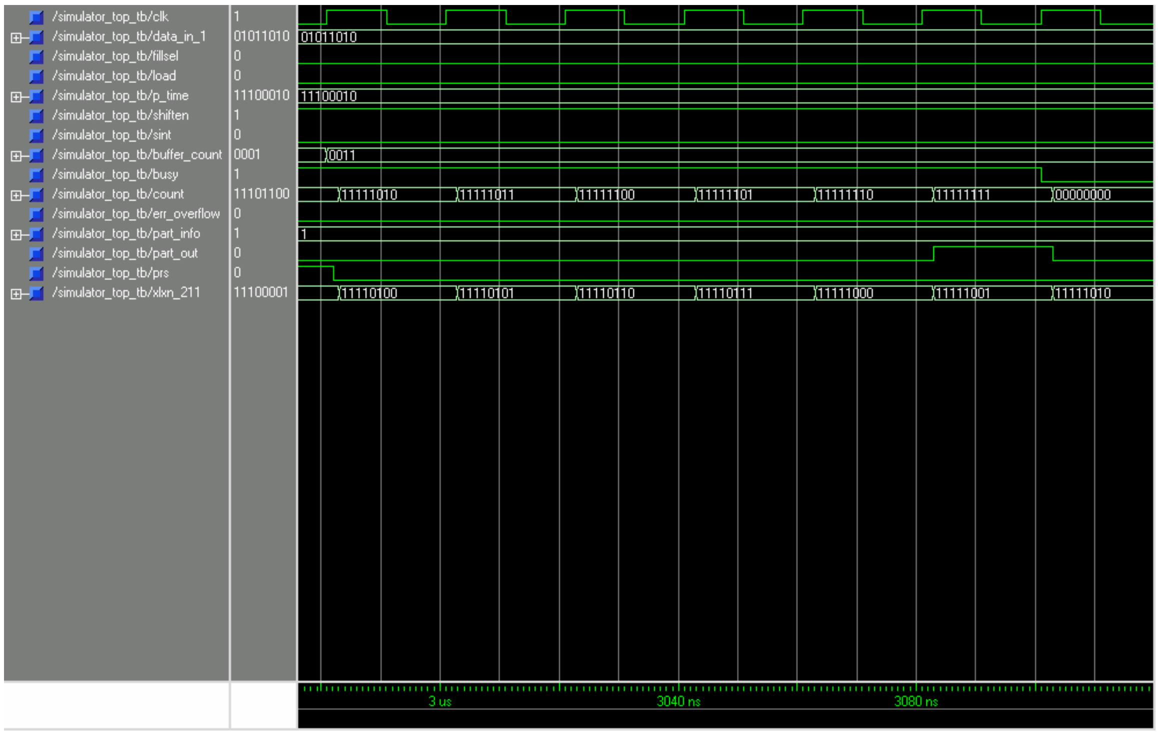
One-Machine-1-Part unit waveform


/simulator_top_tb/clk 1
/simulator_top_tb/da... 01011010
/simulator_top_tb/fillsel 0
/simulator_top_tb/load 0
/simulator_top_tb/p_... 11100010
/simulator_top_tb/shi... 1
/simulator_top_tb/sint 0
/simulator_top_tb/bu... 0001
/simulator_top_tb/busy 1
/simulator_top_tb/co... 11101100
/simulator_top_tb/err... 0
/simulator_top_tb/pa... 1
/simulator_top_tb/pa... 0
/simulator_top_tb/prs 0
/simulator_top_tb/xlx... 11100001
28 us
32 us
36 us
40 us
44 us

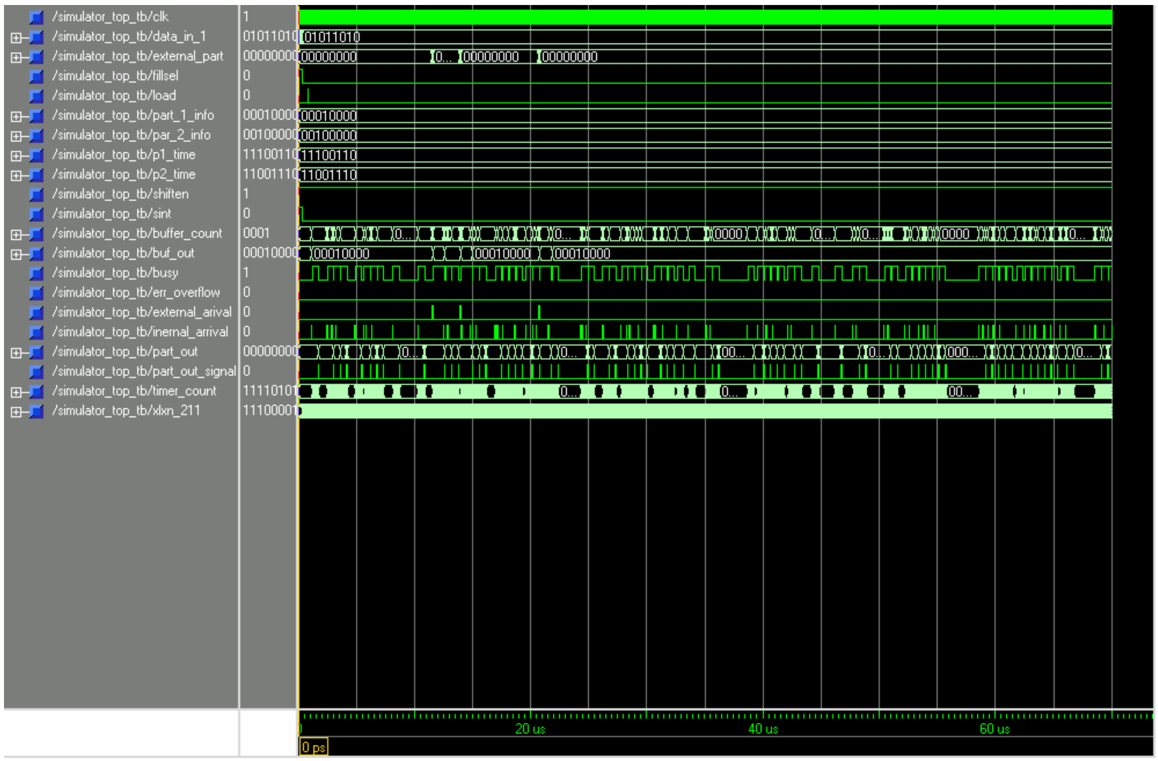
One-Machine-1-Part (2): Close-up view


/simulator_top_tb/clk 1
/simulator_top_tb/da... 01011010
/simulator_top_tb/fillsel 0
/simulator_top_tb/load 0
/simulator_top_tb/p_... 11100010
/simulator_top_tb/shi... 1
/simulator_top_tb/sint 0
/simulator_top_tb/bu... 0001
/simulator_top_tb/busy 1
/simulator_top_tb/co... 11101100
/simulator_top_tb/err... 0
/simulator_top_tb/pa... 1
/simulator_top_tb/pa... 0
/simulator_top_tb/prs 0
/simulator_top_tb/xlx... 11100001
35 us
36 us
37 us
38 us
39 us

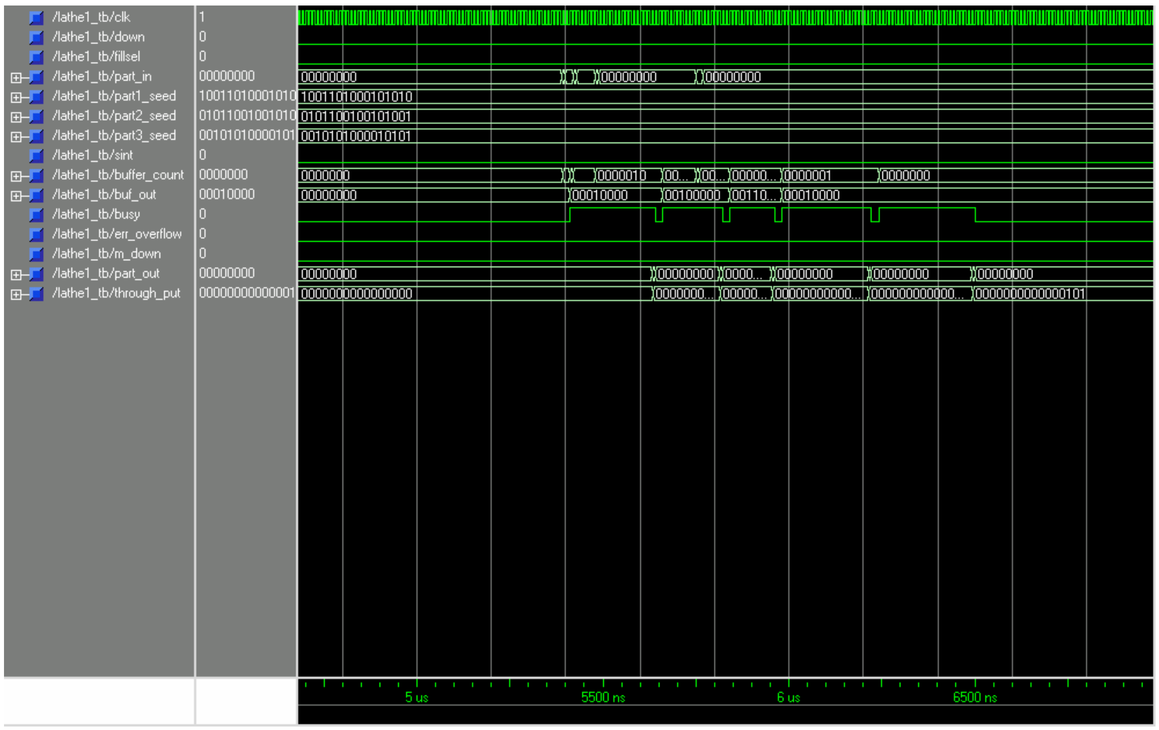
One-Machine-1-Part (3): Close-up view


/simulator_top_tb/clk 1
/simulator_top_tb/data_in_1 01011010
/simulator_top_tb/fillsel 0
/simulator_top_tb/load 0
/simulator_top_tb/p_time 11100010
/simulator_top_tb/shiften 1
/simulator_top_tb/sint 0
/simulator_top_tb/buffer_count 0001
/simulator_top_tb/busy 1
/simulator_top_tb/count 11101100
/simulator_top_tb/err_overflow 0
/simulator_top_tb/part_info 1
/simulator_top_tb/part_out 0
/simulator_top_tb/prs 0
/simulator_top_tb/xlxn_211 11100001
01011010
11100010
0011
11111010
11111011
11111100
11111101
11111110
11111111
00000000
1
11110100
11110101
11110110
11110111
11111000
11111001
11111010
3 us
3040 ns
3080 ns

One-Machine-1-Part (4): Close-up view

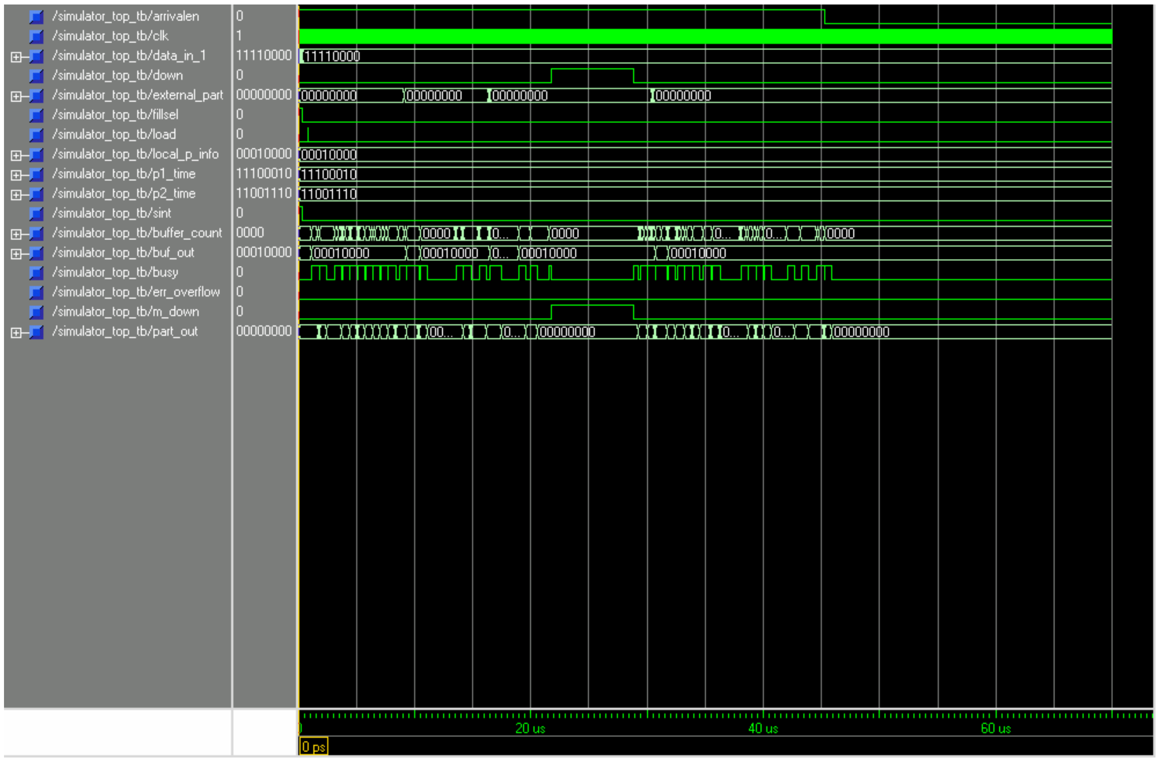

/simulator_top_tb/clk 1
/simulator_top_tb/data_in_1 01011010
/simulator_top_tb/external_part 00000000
/simulator_top_tb/fillsel 0
/simulator_top_tb/load 0
/simulator_top_tb/part_1_info 00010000
/simulator_top_tb/par_2_info 00100000
/simulator_top_tb/p1_time 11100110
/simulator_top_tb/p2_time 11001110
/simulator_top_tb/shiften 1
/simulator_top_tb/sint 0
/simulator_top_tb/buffer_count 0001
/simulator_top_tb/buf_out 00010000
/simulator_top_tb/busy 1
/simulator_top_tb/err_overflow 0
/simulator_top_tb/external_arival 0
/simulator_top_tb/internal_arrival 0
/simulator_top_tb/part_out 00000000
/simulator_top_tb/part_out_signal 0
/simulator_top_tb/timer_count 11110101
/simulator_top_tb/xlxn_211 11100001
01011010
00000000
00010000
00100000
11100110
11001110
0000
00010000
20 us
40 us
60 us
0 ps

One-Machine-Multiple-Part unit waveform



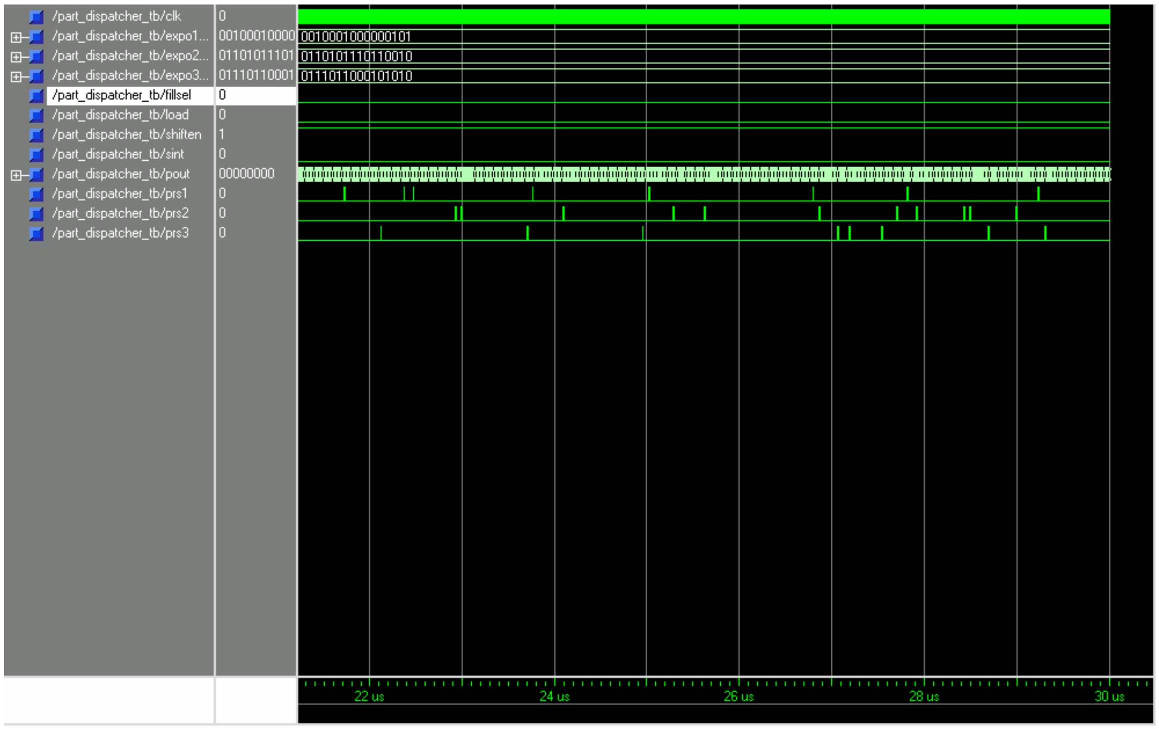
```
/lathe1_tb/clk             1
/lathe1_tb/down            0
/lathe1_tb/fillsel         0
/lathe1_tb/part_in         00000000
/lathe1_tb/part1_seed      10011010001010
/lathe1_tb/part2_seed      01011001001010
/lathe1_tb/part3_seed      00101010000101
/lathe1_tb/sint            0
/lathe1_tb/buffer_count    0000000
/lathe1_tb/buf_out         00010000
/lathe1_tb/busy            0
/lathe1_tb/err_overflow    0
/lathe1_tb/m_down          0
/lathe1_tb/part_out        00000000
/lathe1_tb/through_put     00000000000001
```

One-Machine-Multiple-Part (2)



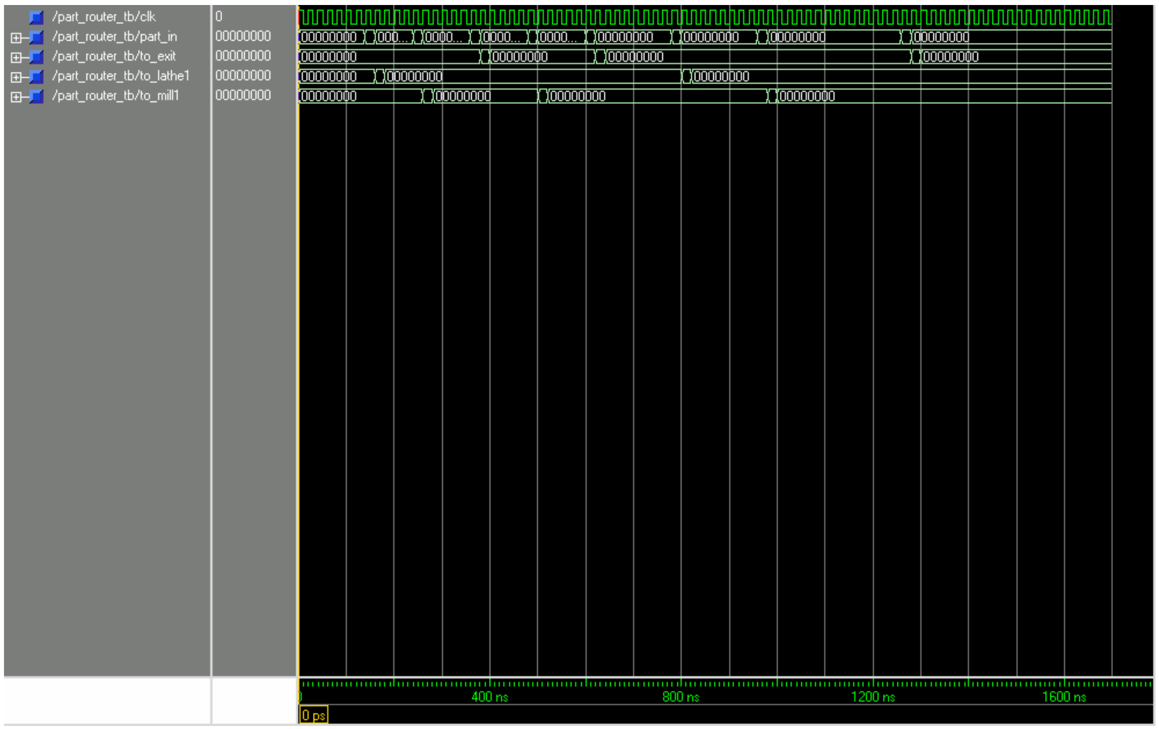
```
/simulator_top_tb/arrivalen       0
/simulator_top_tb/clk             1
/simulator_top_tb/data_in_1       11110000
/simulator_top_tb/down            0
/simulator_top_tb/external_part   00000000
/simulator_top_tb/fillsel         0
/simulator_top_tb/load            0
/simulator_top_tb/local_p_info    00010000
/simulator_top_tb/p1_time         11100010
/simulator_top_tb/p2_time         11001110
/simulator_top_tb/sint            0
/simulator_top_tb/buffer_count    0000
/simulator_top_tb/buf_out         00010000
/simulator_top_tb/busy            0
/simulator_top_tb/err_overflow    0
/simulator_top_tb/m_down          0
/simulator_top_tb/part_out        00000000
```

Down time control waveform



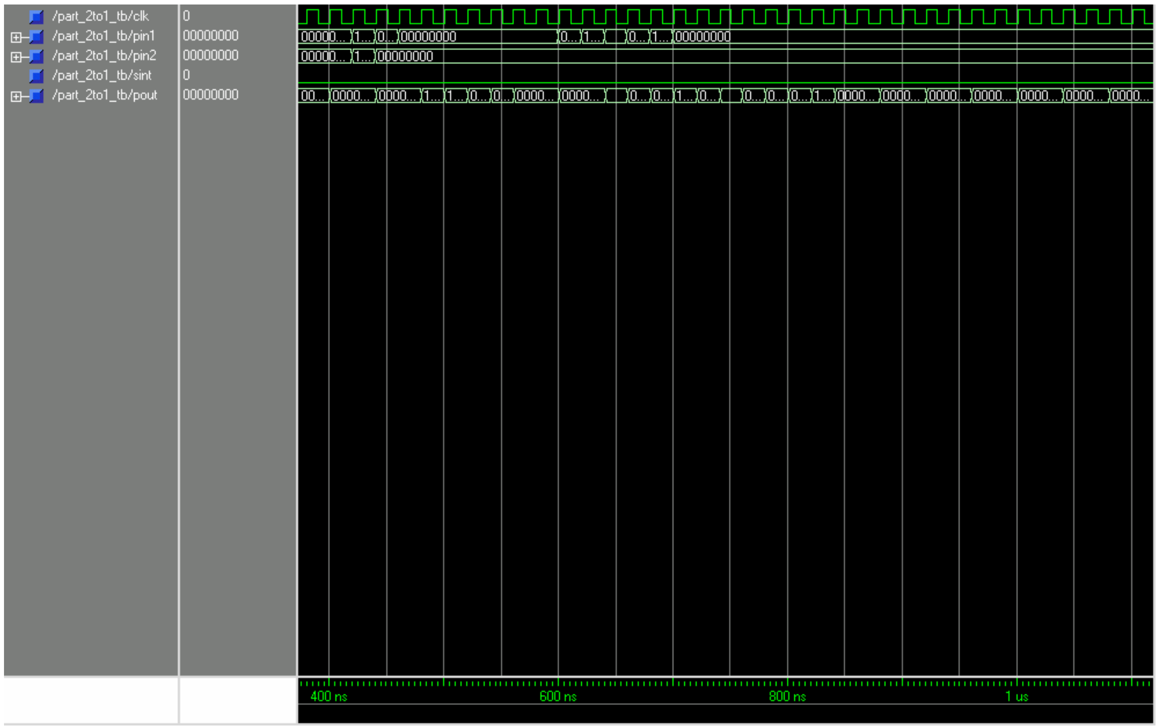
```
/part_dispatcher_tb/clk        0
/part_dispatcher_tb/expo1...   00100010000   0010001000000101
/part_dispatcher_tb/expo2...   01101011101   0110101110110010
/part_dispatcher_tb/expo3...   01110110001   0111011000101010
/part_dispatcher_tb/fillsel    0
/part_dispatcher_tb/load       0
/part_dispatcher_tb/shiften    1
/part_dispatcher_tb/sint       0
/part_dispatcher_tb/pout       00000000
/part_dispatcher_tb/prs1       0
/part_dispatcher_tb/prs2       0
/part_dispatcher_tb/prs3       0
```

Part Dispatcher waveform

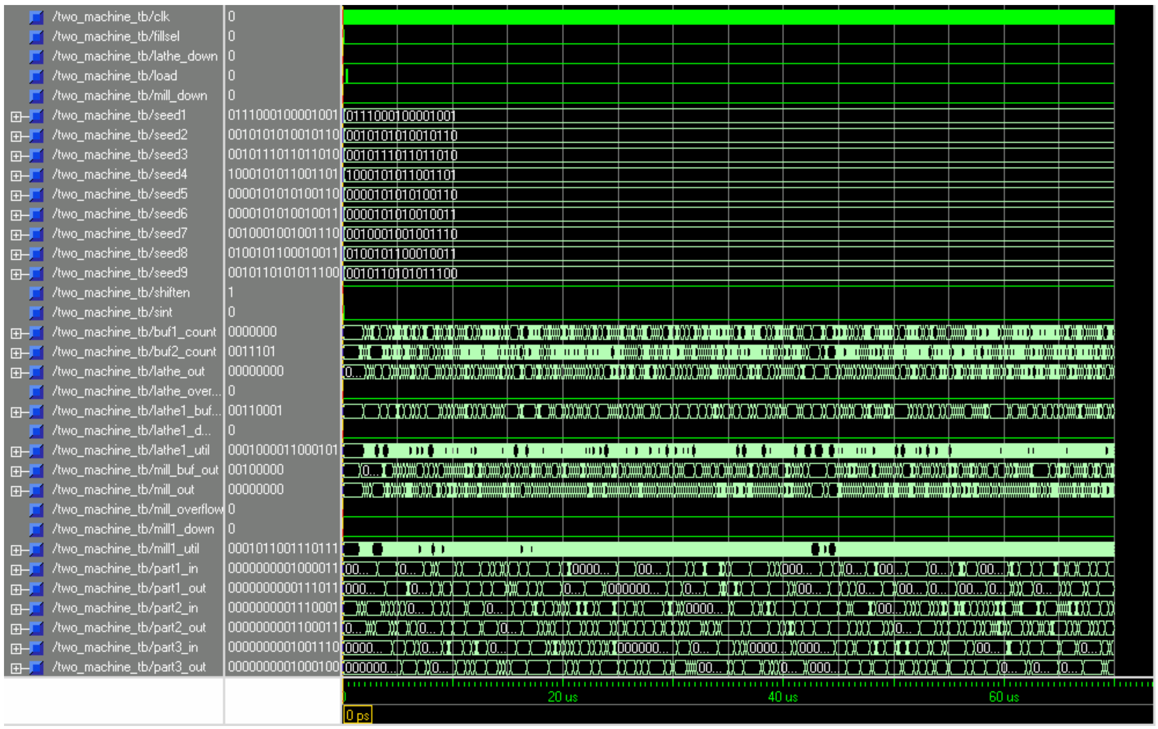


```
/part_router_tb/clk         0
/part_router_tb/part_in     00000000
/part_router_tb/to_exit     00000000
/part_router_tb/to_lathe1   00000000
/part_router_tb/to_mill1    00000000
```

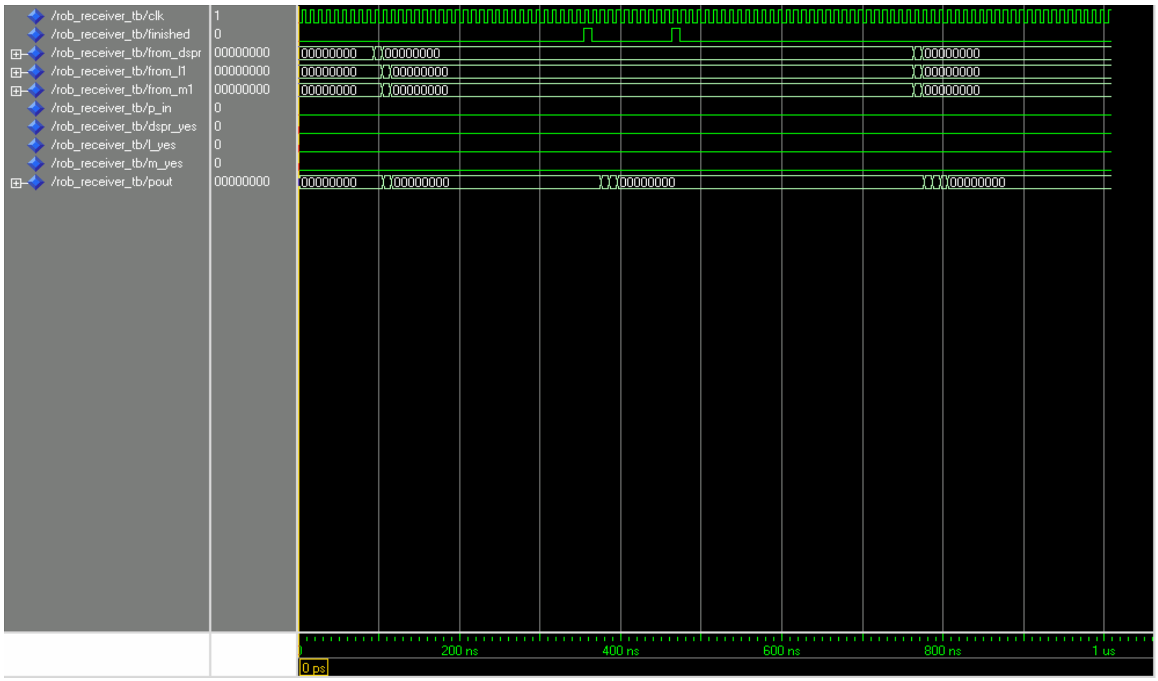
Part Router waveform


/part_2to1_tb/clk 0
/part_2to1_tb/pin1 00000000
/part_2to1_tb/pin2 00000000
/part_2to1_tb/sint 0
/part_2to1_tb/pout 00000000
400 ns
600 ns
800 ns
1 us

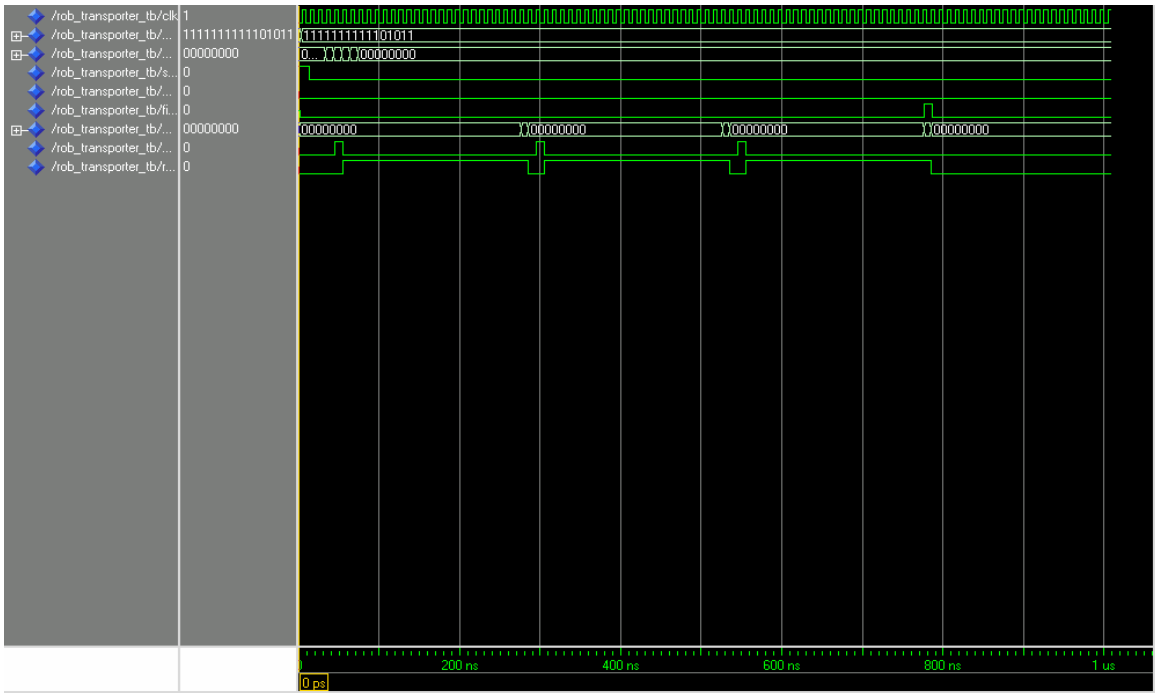
Part_2to1 waveform


/two_machine_tb/clk 0
/two_machine_tb/fillsel 0
/two_machine_tb/lathe_down 0
/two_machine_tb/load 0
/two_machine_tb/mill_down 0
/two_machine_tb/seed1 0111000100001001
/two_machine_tb/seed2 0010101010010110
/two_machine_tb/seed3 0010111011011010
/two_machine_tb/seed4 1000101011001101
/two_machine_tb/seed5 0000101010100110
/two_machine_tb/seed6 0000101010010011
/two_machine_tb/seed7 0010001001001110
/two_machine_tb/seed8 0100101100010011
/two_machine_tb/seed9 0010110101011100
/two_machine_tb/shiften 1
/two_machine_tb/sint 0
/two_machine_tb/buf1_count 0000000
/two_machine_tb/buf2_count 0011101
/two_machine_tb/lathe_out 00000000
/two_machine_tb/lathe_over... 0
/two_machine_tb/lathe1_buf... 00110001
/two_machine_tb/lathe1_d... 0
/two_machine_tb/lathe1_util 0001000011000101
/two_machine_tb/mill_buf_out 00100000
/two_machine_tb/mill_out 00000000
/two_machine_tb/mill_overflow 0
/two_machine_tb/mill1_down 0
/two_machine_tb/mill1_util 0001011001110111
/two_machine_tb/part1_in 0000000001000011
/two_machine_tb/part1_out 0000000000111011
/two_machine_tb/part2_in 0000000001110001
/two_machine_tb/part2_out 0000000001100011
/two_machine_tb/part3_in 0000000001001110
/two_machine_tb/part3_out 0000000001000100
20 us
40 us
60 us
0 ps

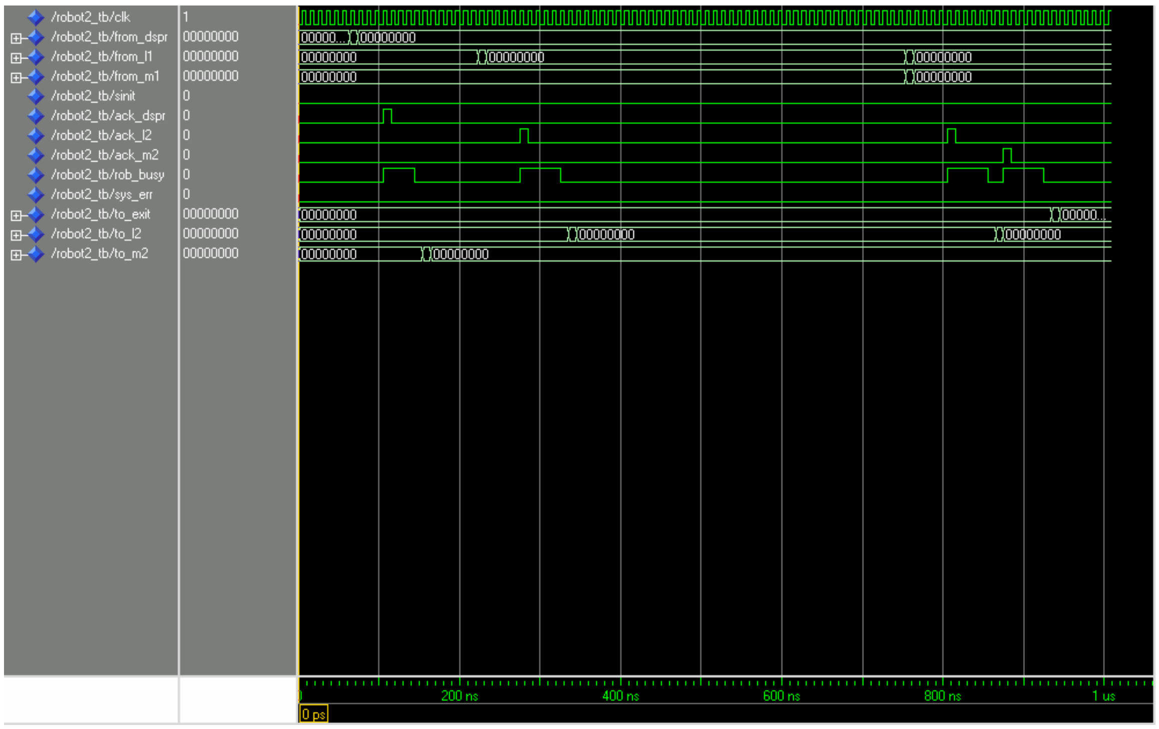
Two-Machine-3-part cell waveform


/rob_receiver_tb/clk 1
/rob_receiver_tb/finished 0
/rob_receiver_tb/from_dspr 00000000
/rob_receiver_tb/from_l1 00000000
/rob_receiver_tb/from_m1 00000000
/rob_receiver_tb/p_in 0
/rob_receiver_tb/dspr_yes 0
/rob_receiver_tb/l_yes 0
/rob_receiver_tb/m_yes 0
/rob_receiver_tb/pout 00000000
200 ns
400 ns
600 ns
800 ns
1 us
0 ps

Robot Receiver signal waveform

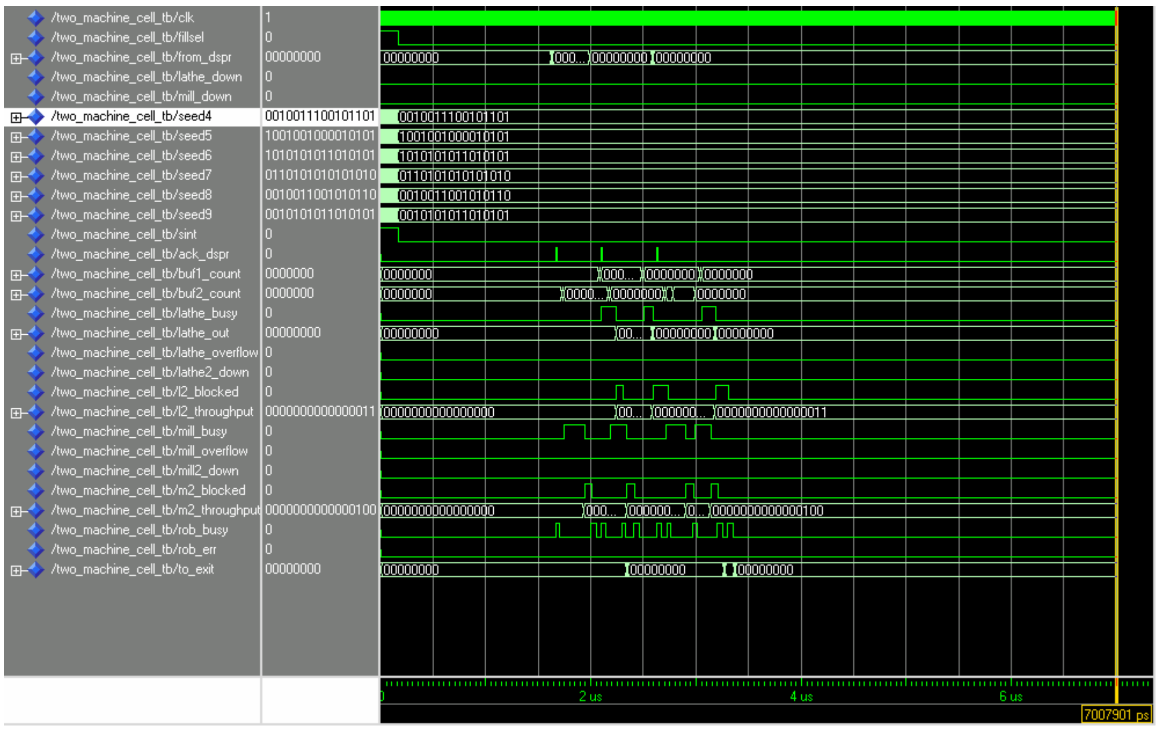

/rob_transporter_tb/clk 1
/rob_transporter_tb/... 1111111111101011
/rob_transporter_tb/... 00000000
/rob_transporter_tb/s... 0
/rob_transporter_tb/... 0
/rob_transporter_tb/fi... 0
/rob_transporter_tb/... 00000000
/rob_transporter_tb/... 0
/rob_transporter_tb/r... 0
200 ns
400 ns
600 ns
800 ns
1 us
0 ps

Robot Transporter signal waveform



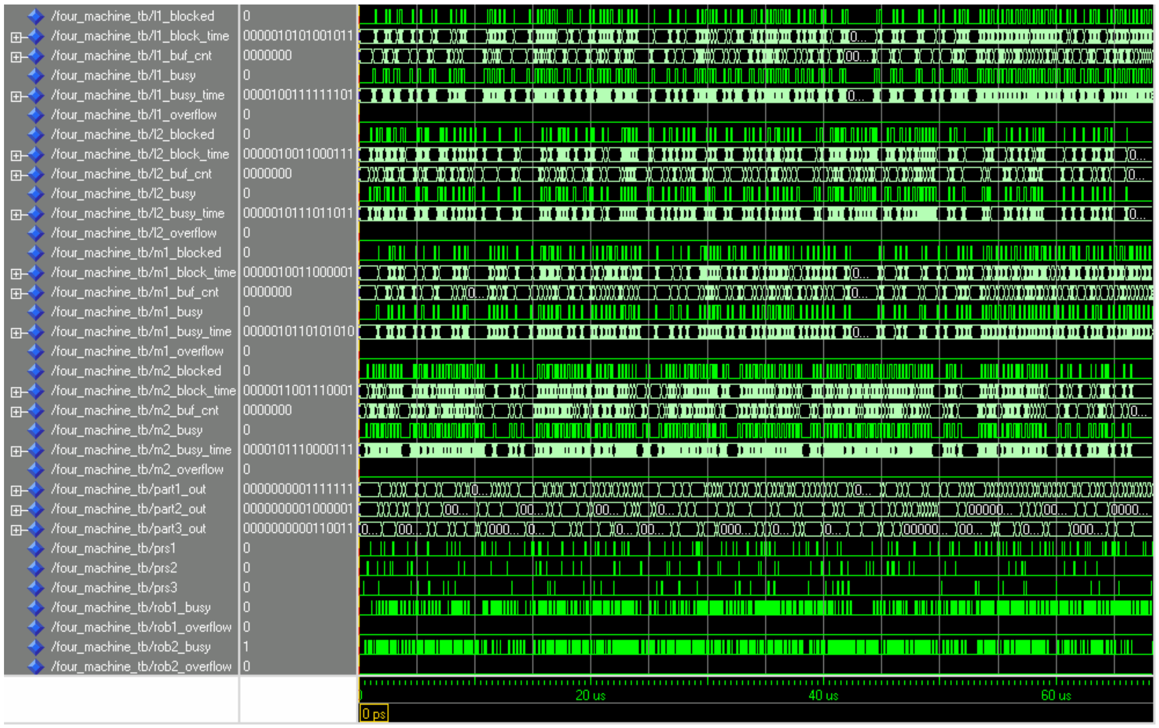
```
/robot2_tb/clk          1
/robot2_tb/from_dspr    00000000
/robot2_tb/from_l1      00000000
/robot2_tb/from_m1      00000000
/robot2_tb/sinit        0
/robot2_tb/ack_dspr     0
/robot2_tb/ack_l2       0
/robot2_tb/ack_m2       0
/robot2_tb/rob_busy     0
/robot2_tb/sys_err      0
/robot2_tb/to_exit      00000000
/robot2_tb/to_l2        00000000
/robot2_tb/to_m2        00000000
```

Robot waveform



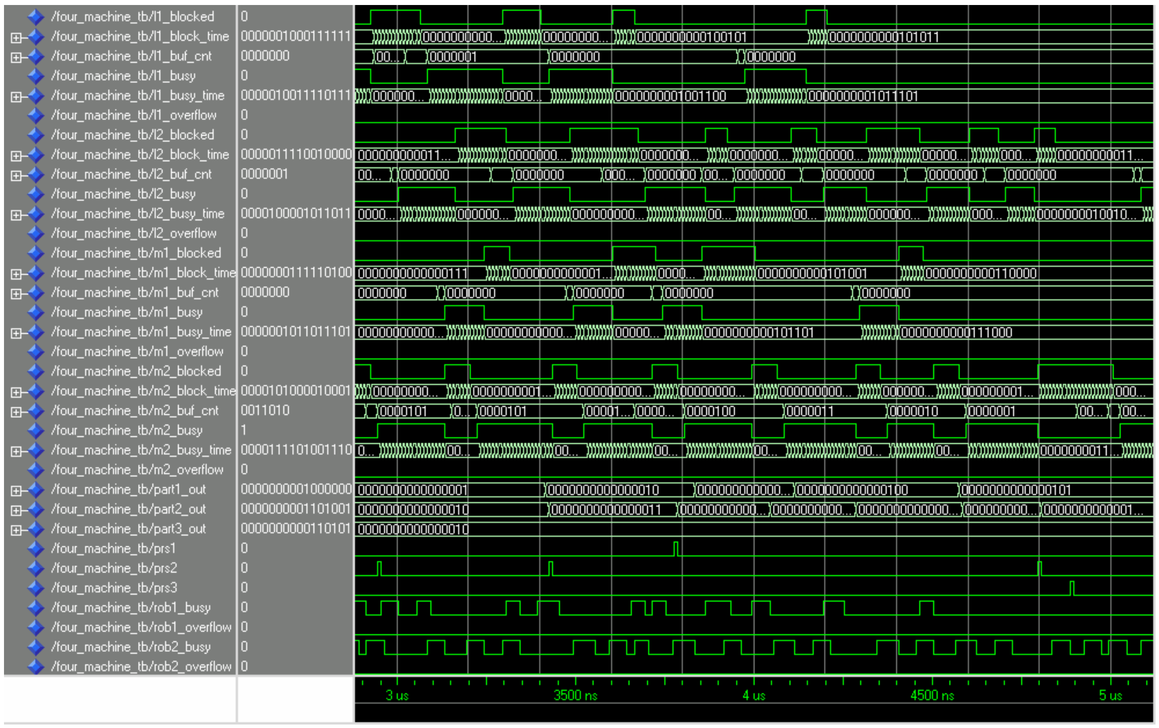
```
/two_machine_cell_tb/clk              1
/two_machine_cell_tb/fillsel          0
/two_machine_cell_tb/from_dspr        00000000
/two_machine_cell_tb/lathe_down       0
/two_machine_cell_tb/mill_down        0
/two_machine_cell_tb/seed4            0010011100101101
/two_machine_cell_tb/seed5            1001001000010101
/two_machine_cell_tb/seed6            1010101011010101
/two_machine_cell_tb/seed7            0110101010101010
/two_machine_cell_tb/seed8            0010011001010110
/two_machine_cell_tb/seed9            0010101011010101
/two_machine_cell_tb/sint             0
/two_machine_cell_tb/ack_dspr         0
/two_machine_cell_tb/buf1_count       0000000
/two_machine_cell_tb/buf2_count       0000000
/two_machine_cell_tb/lathe_busy       0
/two_machine_cell_tb/lathe_out        00000000
/two_machine_cell_tb/lathe_overflow   0
/two_machine_cell_tb/lathe2_down      0
/two_machine_cell_tb/l2_blocked       0
/two_machine_cell_tb/l2_throughput    0000000000000011
/two_machine_cell_tb/mill_busy        0
/two_machine_cell_tb/mill_overflow    0
/two_machine_cell_tb/mill2_down       0
/two_machine_cell_tb/m2_blocked       0
/two_machine_cell_tb/m2_throughput    0000000000000100
/two_machine_cell_tb/rob_busy         0
/two_machine_cell_tb/rob_err          0
/two_machine_cell_tb/to_exit          00000000
```

Two-Machine cell with robot waveform



```
/four_machine_tb/l1_blocked         0
/four_machine_tb/l1_block_time      0000010101001011
/four_machine_tb/l1_buf_cnt         0000000
/four_machine_tb/l1_busy            0
/four_machine_tb/l1_busy_time       0000100111111101
/four_machine_tb/l1_overflow        0
/four_machine_tb/l2_blocked         0
/four_machine_tb/l2_block_time      0000010011000111
/four_machine_tb/l2_buf_cnt         0000000
/four_machine_tb/l2_busy            0
/four_machine_tb/l2_busy_time       0000010111011011
/four_machine_tb/l2_overflow        0
/four_machine_tb/m1_blocked         0
/four_machine_tb/m1_block_time      0000010011000001
/four_machine_tb/m1_buf_cnt         0000000
/four_machine_tb/m1_busy            0
/four_machine_tb/m1_busy_time       0000010110101010
/four_machine_tb/m1_overflow        0
/four_machine_tb/m2_blocked         0
/four_machine_tb/m2_block_time      0000011001110001
/four_machine_tb/m2_buf_cnt         0000000
/four_machine_tb/m2_busy            0
/four_machine_tb/m2_busy_time       0000101110000111
/four_machine_tb/m2_overflow        0
/four_machine_tb/part1_out          0000000001111111
/four_machine_tb/part2_out          0000000001000001
/four_machine_tb/part3_out          0000000000110011
/four_machine_tb/prs1               0
/four_machine_tb/prs2               0
/four_machine_tb/prs3               0
/four_machine_tb/rob1_busy          0
/four_machine_tb/rob1_overflow      0
/four_machine_tb/rob2_busy          1
/four_machine_tb/rob2_overflow      0
```

Four-Machine FMS waveform

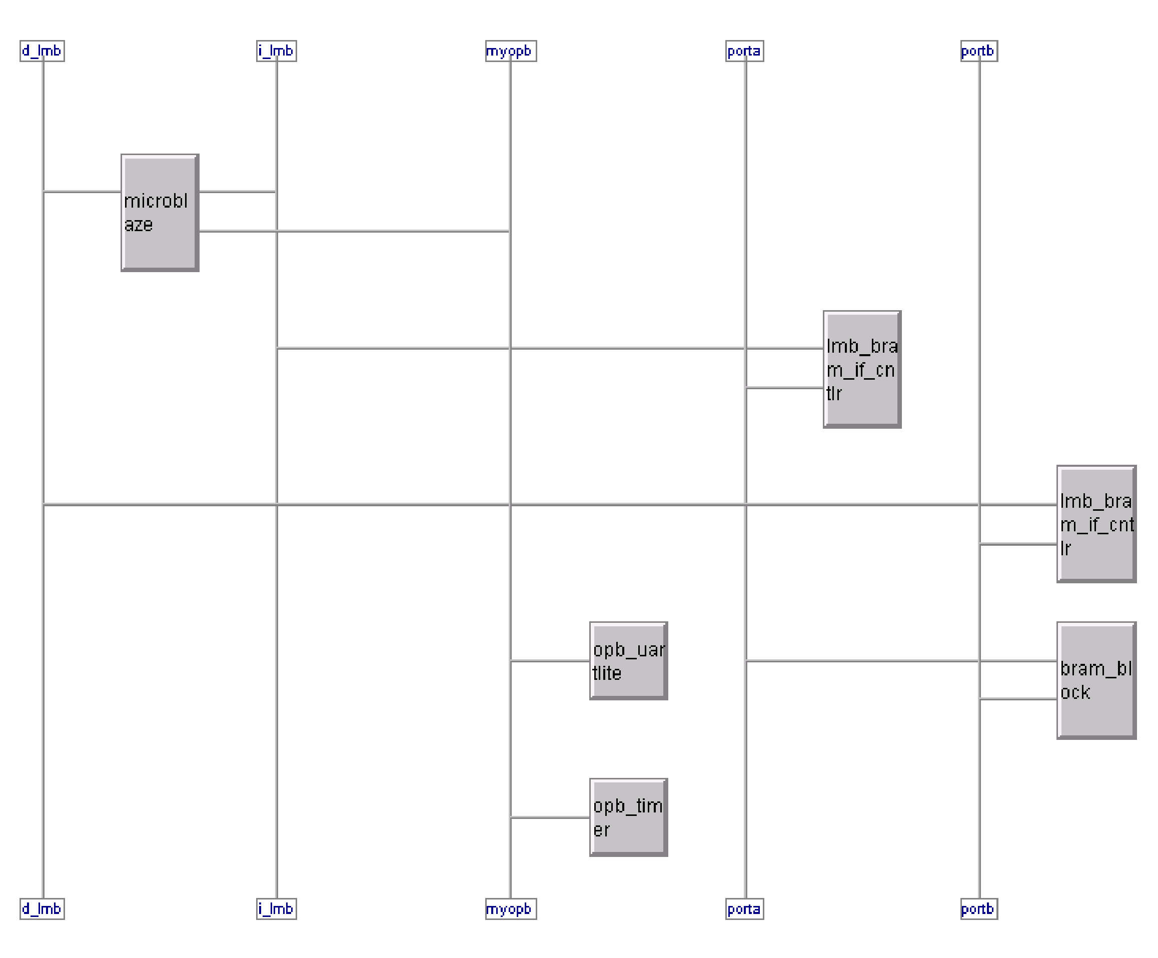


```
/four_machine_tb/l1_blocked         0
/four_machine_tb/l1_block_time      0000001000111111
/four_machine_tb/l1_buf_cnt         0000000
/four_machine_tb/l1_busy            0
/four_machine_tb/l1_busy_time       0000010011110111
/four_machine_tb/l1_overflow        0
/four_machine_tb/l2_blocked         0
/four_machine_tb/l2_block_time      0000011110010000
/four_machine_tb/l2_buf_cnt         0000001
/four_machine_tb/l2_busy            0
/four_machine_tb/l2_busy_time       0001000001011011
/four_machine_tb/l2_overflow        0
/four_machine_tb/m1_blocked         0
/four_machine_tb/m1_block_time      0000000111110100
/four_machine_tb/m1_buf_cnt         0000000
/four_machine_tb/m1_busy            0
/four_machine_tb/m1_busy_time       0000001011011101
/four_machine_tb/m1_overflow        0
/four_machine_tb/m2_blocked         0
/four_machine_tb/m2_block_time      0001010000010001
/four_machine_tb/m2_buf_cnt         0011010
/four_machine_tb/m2_busy            1
/four_machine_tb/m2_busy_time       0000111101001110
/four_machine_tb/m2_overflow        0
/four_machine_tb/part1_out          0000000001000000
/four_machine_tb/part2_out          0000000001101001
/four_machine_tb/part3_out          0000000000110101
/four_machine_tb/prs1               0
/four_machine_tb/prs2               0
/four_machine_tb/prs3               0
/four_machine_tb/rob1_busy          0
/four_machine_tb/rob1_overflow      0
/four_machine_tb/rob2_busy          0
/four_machine_tb/rob2_overflow      0
```

Four-Machine FMS waveform (2) --- Close up view

APPENDIX C: Program Documentation

MicroBlaze DES Model



System architecture -- PBD file

Application file --- 1-machine.c

```
/* 1-machine.c -- Discrete event simulation of one machine center.
 * This is the program running on a Xilinx MicroBlaze softcore microprocessor based on
 * Virtex-II FPGA development kit.
 * Programmed by: Wei Tang
 * Date: May, 2003
```

```
*/
#include "math.h"
#include "xuartlite_l.h"
#include "xtmrctr_l.h"
#include "xparameters.h"
#include "stdio.h"

#define QLimit 20        /*Number of part type*/
#define WIPLimit 5        /*Limitation of WIP queue*/
#define NumEvet 2         /*Number of event*/
#define NumPType 2        /*Number of part type*/

struct Part {
        int P_Type;     /*Part type = 0(Part A), 1(Part B)*/
        int P_Task[2];  /*Operation required on 2 machines, 0(No), 1(Yes)*/
        float P_Enter;  /*Time when entering the FMS*/
        float P_Start;  /*Arrival time to one workstation*/
        float P_Finish; /*Finish time of one operation on one workstation*/
        };

struct Event {
        int E_Type;     /*Event type = 1(Arrival), 2(Departure), 3(Machine Broken)*/
        int P_Type;     /*Which part related to the event, 0(Part A), 1(Part B)*/
        int M_Type;     /*Which machine related to the event, 1(Workstation 1), 2(Workstation 2)*/
        float E_Time;   /*Event happen time*/
        };

struct Machine {
        int M_Type;     /*Machine type, 1 = lathe, 2 = mill*/
        int M_State;    /*Machine state = 0(idle), 1(busy), 2(blocked), 3(broken)*/
        int P_Type;     /*Current operating part type*/
        float P_Time[2];/*Processing time each part*/
        };

/* Define global variables*/
int NxtEvet=0;          /*Next event type, 0=empty, 1=arrival, 2=departure, 3=machine broken*/
int NumInQ=0;           /*Number of parts waiting in M1 input queue*/
int TotNum=0;           /*Total number of part entering*/

/* Used to count the part throughput */
int SumPartA=0;         /*Total finished Part A*/

float SumInQ=0.0;       /*Sum of time of total parts in queue*/
float SumM1=0.0;        /*Sum of machine 1 utilization*/
float SumDelay=0.0;     /*Total delay time of parts*/
float SumBlock=0.0;     /*Total block time of parts*/
float SumStay=0.0;      /*Total time stay in system*/

float Time=0.0;         /*Current simulating time, in minutes*/
float LastTime=0.0;     /*Time of last event*/
float EndTime=7000.0;  /*End simulation time, 50 hours*/
static float MeArriA=45.0;      /*Mean time between arrival of Part A*/

/* These global variables are used to pass parameters between functions to execute the
simulation,
 * In this way, it is easy to compile this C program to assemble language*/
struct Machine M1;      /*Describe this workstation*/
struct Part  CurPart;   /*Current part scheduled to be processed on machine 1*/
struct Event EveList[NumEvet+1];/*Event list of this program*/
struct Part  InputQ[QLimit+1]; /*Input queue for the workstation*/

/* Prototype of functions */
void initial(void);
void timing(void);
void arrive(void);
void depart(void);
void report(void);
float expon(float mean);
float UniformRandNum(void);

main() {  unsigned long int timer_data;

  print("program starts.\n\r");

    /* set initial value to 0 */
  XTmrCtr_mSetLoadReg(XPAR_MYTIMER_BASEADDR, 0, 0);

  /* reset the timers */
  XTmrCtr_mSetControlStatusReg(XPAR_MYTIMER_BASEADDR,                                  0,
XTC_CSR_LOAD_MASK );

  /* start the timer, count up*/
  XTmrCtr_mSetControlStatusReg(XPAR_MYTIMER_BASEADDR,                                  0,
XTC_CSR_ENABLE_TMR_MASK);

  initial();
  while(Time <= EndTime) {

        /*Determin the next event type*/
        timing();

        /*Invoke the appropriate event handle function*/
        switch(NxtEvet){
           case 1:
              arrive();   /*Handle Part arrival event*/
              break;
           case 2:
              depart();   /*Handle Part departure event*/
              break;
        }
    }
     /* stop the timer */
  XTmrCtr_mSetControlStatusReg(XPAR_MYTIMER_BASEADDR, 0, 0);

  /* read register value from the timer */
  timer_data = XTmrCtr_mGetTimerCounterReg(XPAR_MYTIMER_BASEADDR, 0);

  xil_printf("program execution cycle time: %d\n\r", timer_data);

           report();              /*Simulation results report*/
    xil_printf("program finished normally\n\r");
    return 0;
}

/*----------------------------------------------------------------------
 *Initilization function, schedule the first arrival
 */
void initial(void)
{ register int i,j;
  /*Global variables has been initialized when declare*/
    TotNum = 1;
  /*Describe the workstation 1*/
    M1.M_Type = 1;          /*Workstation 1 is a lathe*/
    M1.M_State = 0;         /*Workstation 1 initial state is idle*/
    M1.P_Type = 0;          /*Current part type on workstation 1 is Part A*/
    M1.P_Time[0] = 20.0;     /*Processing time of Part A on workstation 1 is 30.0 minutes*/
    M1.P_Time[1] = 0.0;     /*Not availabe to process Part B*/
    /*Describe the next part scheduled on workstation 1, InputQ[0] is the storage of next part*/
    InputQ[0].P_Type = 0;     /*Part A*/
    InputQ[0].P_Task[0] = 1;  /*Part A need turning operation on workstation 1*/
    InputQ[0].P_Task[1] = 1;  /*Part A need drilling operation on workstation 2*/
    InputQ[0].P_Enter = expon(MeArriA); /*Part A entering FMS at arrival time*/
    InputQ[0].P_Start = 0.0;      /*Part A start on workstation 1 */
    InputQ[0].P_Finish = 0.0;     /*Time to finish operation on workstaion 1 does not known yet*/

    for(i=0;i<=NumEvet;i++){
        EveList[i].E_Type = i;
        EveList[i].P_Type = 0;
        EveList[i].M_Type = 1;
    }

    EveList[1].E_Time = InputQ[0].P_Enter; /*Schedule the arrive time*/
    EveList[2].E_Time = 1.0e+30;    /*Schedule the initial departure time to be infinite*/

    }
void timing(void)
{
    register int i,j;
    register float min_time = 1.0e+29;
    register float time_since_last_event;
    static float q;

    NxtEvet = 0;
    for(i=1;i<=NumEvet;i++){
        q=EveList[i].E_Time-min_time;    /*for testing*/
        if(q<0){
            min_time = EveList[i].E_Time;
            NxtEvet = i;
        }
    }
    if(NxtEvet == 0){
        printf("\nEvent List empty error!");
        exit(1);
    }
    Time = min_time;
    time_since_last_event = Time - LastTime;
    LastTime = Time;

          for(j=1;j<=NumInQ;j++)
      SumInQ += time_since_last_event;
 }
void arrive(void)
{
      /*Check to see if workstation 1 is idle*/
    if(M1.M_State == 0){         /*Machine 1  idle*/
        M1.M_State = 1;          /*Set Machine 1 busy*/
        CurPart.P_Enter = Time;
        CurPart.P_Start = Time;
        /*Schedule a Part 1 departure after processing, considering the travel time of robot*/
       EveList[2].E_Time = CurPart.P_Start + expon(M1.P_Time[0]);
       CurPart.P_Finish = EveList[2].E_Time;
        /*Update Machine busy period*/
        SumM1 += CurPart.P_Finish - CurPart.P_Start;
        }
```

```
   else{
        ++NumInQ;
       if(NumInQ > QLimit){
          printf("\nMachine 1: input Queue overflow error!");
          exit(2);
       }
       /*Leave the arrival part in the queue*/
       InputQ[NumInQ] = InputQ[0];
   }
   InputQ[0].P_Enter = Time + expon(MeArriA);
   EveList[1].E_Time = InputQ[0].P_Enter;
   ++TotNum;
}
void depart(void)
{
   register int i;
   register float delay;
          SumStay += CurPart.P_Finish - CurPart.P_Enter;
          ++SumPartA;
          if(NumInQ == 0){
             /*The input queue is empty, set machine 1 idle*/
             M1.M_State = 0;
             EveList[2].E_Time = 1.0e+30;
             }
          else{
             M1.M_State = 1;
             --NumInQ;
             delay = Time - InputQ[1].P_Enter;
             SumDelay += delay;
             CurPart = InputQ[1];

             for(i=1;i<=NumInQ;i++){
                InputQ[i] = InputQ[i+1];
             }
             CurPart.P_Start = Time;
             EveList[2].E_Time = CurPart.P_Start + expon(M1.P_Time[0]);
             CurPart.P_Finish = EveList[2].E_Time;
             /*Update Machine busy period*/
             SumM1 += CurPart.P_Finish - CurPart.P_Start;
                            }
}
void report(void)
{   static float Ave_delay, Ave_Block, Ave_InQ, Ave_Stay, M1_Util;
    Ave_delay = SumDelay/SumPartA;
   Ave_Block = SumBlock/Time;
   Ave_InQ = SumInQ/Time;
   Ave_Stay = SumStay/SumPartA;
   M1_Util = (long)SumM1*100/(long)Time;

   xil_printf("\nParallel discrete event simulation of an FMS\n\r");
   xil_printf("\nSimulation Time: %d minutes\n\r", (int)EndTime);
   xil_printf("\nTotal part entering: %d\n\r", TotNum);
   xil_printf("\nPart A Throughput: %d\n\r", (int)SumPartA);
   xil_printf("\nAverage delay in queue: %d\n\r", (int)Ave_delay);
   xil_printf("\nAverage number in queue: %d\n\r", (int)Ave_InQ);
   xil_printf("\nAverage Time Part A stays in FMS: %d\n\r", (int)Ave_Stay);
   xil_printf("\nUtilization of Workstation 1: %d percent\n\r", (int)M1_Util);
   xil_printf("\nPercentage of Workstation 1 blocked: %d\n\r", (int)Ave_Block);
}
float expon(float mean) /*Exponential variate generation function*/
{   register float u;
   /*Generate a uniform (0,1) variate*/
   u = UniformRandNum();
   return -mean*log(u);
}
float UniformRandNum(void)
{    static long r = 30000;
     long  m = 65535;
     int a = 1145;
     int c = 13107;
     r = (a*r+c)%m+1;
     return (float)r/m;
}
```

Custom IP integration

System MHS file:

```
PARAMETER VERSION = 2.1.0

 PORT fpga_0_RS232_RX_pin = fpga_0_RS232_RX, DIR = INPUT
 PORT fpga_0_RS232_TX_pin = fpga_0_RS232_TX, DIR = OUTPUT
 PORT fpga_0_RS232_req_to_send_pin = net_gnd, DIR = OUTPUT
 PORT sys_clk_pin = dcm_clk_s, DIR = INPUT, SIGIS = DCMCLK
 PORT sys_rst_pin = sys_rst_s, DIR = INPUT

BEGIN microblaze
 PARAMETER INSTANCE = microblaze_0
 PARAMETER HW_VER = 4.00.a
 PARAMETER C_DEBUG_ENABLED = 1
 PARAMETER C_NUMBER_OF_PC_BRK = 2
 PARAMETER C_NUMBER_OF_RD_ADDR_BRK = 1
 PARAMETER C_NUMBER_OF_WR_ADDR_BRK = 1
 BUS_INTERFACE DLMB = dlmb
 BUS_INTERFACE ILMB = ilmb
 BUS_INTERFACE DOPB = mb_opb
 BUS_INTERFACE IOPB = mb_opb
 PORT CLK = sys_clk_s
 PORT DBG_CAPTURE = DBG_CAPTURE_s
 PORT DBG_CLK = DBG_CLK_s
 PORT DBG_REG_EN = DBG_REG_EN_s
 PORT DBG_TDI = DBG_TDI_s
 PORT DBG_TDO = DBG_TDO_s
 PORT DBG_UPDATE = DBG_UPDATE_s
END

BEGIN opb_v20
 PARAMETER INSTANCE = mb_opb
 PARAMETER HW_VER = 1.10.c
 PARAMETER C_EXT_RESET_HIGH = 0
 PORT SYS_Rst = sys_rst_s
 PORT OPB_Clk = sys_clk_s
END

BEGIN opb_mdm
 PARAMETER INSTANCE = debug_module
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_MB_DBG_PORTS = 1
 PARAMETER C_USE_UART = 1
 PARAMETER C_UART_WIDTH = 8
 PARAMETER C_BASEADDR = 0x41400000
 PARAMETER C_HIGHADDR = 0x4140ffff
 BUS_INTERFACE SOPB = mb_opb
 PORT OPB_Clk = sys_clk_s
 PORT DBG_CAPTURE_0 = DBG_CAPTURE_s
 PORT DBG_CLK_0 = DBG_CLK_s
 PORT DBG_REG_EN_0 = DBG_REG_EN_s
 PORT DBG_TDI_0 = DBG_TDI_s
 PORT DBG_TDO_0 = DBG_TDO_s
 PORT DBG_UPDATE_0 = DBG_UPDATE_s
END

BEGIN lmb_v10
 PARAMETER INSTANCE = ilmb
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_EXT_RESET_HIGH = 0
 PORT SYS_Rst = sys_rst_s
 PORT LMB_Clk = sys_clk_s
```

```
END

BEGIN lmb_v10
 PARAMETER INSTANCE = dlmb
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_EXT_RESET_HIGH = 0
 PORT SYS_Rst = sys_rst_s
 PORT LMB_Clk = sys_clk_s
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = dlmb_cntlr
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x0000ffff
 BUS_INTERFACE SLMB = dlmb
 BUS_INTERFACE BRAM_PORT = dlmb_port
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = ilmb_cntlr
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x0000ffff
 BUS_INTERFACE SLMB = ilmb
 BUS_INTERFACE BRAM_PORT = ilmb_port
END

BEGIN bram_block
 PARAMETER INSTANCE = lmb_bram
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA = ilmb_port
 BUS_INTERFACE PORTB = dlmb_port
END

BEGIN opb_uartlite
 PARAMETER INSTANCE = RS232
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_BAUDRATE = 19200
 PARAMETER C_DATA_BITS = 8
 PARAMETER C_ODD_PARITY = 0
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_CLK_FREQ = 100000000
 PARAMETER C_BASEADDR = 0x40600000
 PARAMETER C_HIGHADDR = 0x4060ffff
 BUS_INTERFACE SOPB = mb_opb
 PORT OPB_Clk = sys_clk_s
 PORT RX = fpga_0_RS232_RX
 PORT TX = fpga_0_RS232_TX
END

BEGIN dcm_module
 PARAMETER INSTANCE = dcm_0
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_CLK0_BUF = TRUE
 PARAMETER C_CLKIN_PERIOD = 10.000000
 PARAMETER C_CLK_FEEDBACK = 1X
 PARAMETER C_EXT_RESET_HIGH = 1
 PORT CLKIN = dcm_clk_s
 PORT CLK0 = sys_clk_s
 PORT CLKFB = sys_clk_s
 PORT RST = net_gnd
 PORT LOCKED = dcm_0_lock
END
```

System MSS file:

```
PARAMETER VERSION = 2.2.0

BEGIN OS
 PARAMETER OS_NAME = standalone
 PARAMETER OS_VER = 1.00.a
 PARAMETER PROC_INSTANCE = microblaze_0
 PARAMETER STDIN = RS232
 PARAMETER STDOUT = RS232
END

BEGIN PROCESSOR
 PARAMETER DRIVER_NAME = cpu
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = microblaze_0
 PARAMETER COMPILER = mb-gcc
 PARAMETER ARCHIVER = mb-ar
 PARAMETER XMDSTUB_PERIPHERAL = debug_module
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = opbarb
 PARAMETER DRIVER_VER = 1.02.a
 PARAMETER HW_INSTANCE = mb_opb
END

BEGIN DRIVER
```

```
 PARAMETER DRIVER_NAME = uartlite
 PARAMETER DRIVER_VER = 1.00.b
 PARAMETER HW_INSTANCE = debug_module
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = bram
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = dlmb_cntlr
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = bram
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = ilmb_cntlr
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = uartlite
 PARAMETER DRIVER_VER = 1.00.b
 PARAMETER HW_INSTANCE = RS232
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = dcm_0
END
```

MB_simulator custom IP VHDL file:

```
------------------------------------------------------------------------------
-- user_logic.vhd - entity/architecture pair
------------------------------------------------------------------------------
--
-- ***************************************************************************
-- ** Programmed by Wei Tang                                          .     **
-- **                                                                       **
-- ** September 2005                                                        **
-- ***************************************************************************
--
------------------------------------------------------------------------------
-- Filename:          user_logic.vhd
-- Version:           1.00.a
-- Description:       User logic.
-- Date:              Sun Oct 09 16:39:31 2005 (by Create and Import Peripheral Wizard)
-- VHDL Standard:     VHDL'93
```

```
------------------------------------------------------------------------------
-- Naming Conventions:
--   active low signals:                    "*_n"
--   clock signals:                         "clk", "clk_div#", "clk_#x"
--   reset signals:                         "rst", "rst_n"
--   generics:                              "C_*"
--   user defined types:                    "*_TYPE"
--   state machine next state:              "*_ns"
--   state machine current state:           "*_cs"
--   combinatorial signals:                 "*_com"
--   pipelined or register delay signals:   "*_d#"
--   counter signals:                       "*cnt*"
--   clock enable signals:                  "*_ce"
--   internal version of output port:       "*_i"
--   device pins:                           "*_pin"
--   ports:                                 "- Names begin with Uppercase"
--   processes:                             "*_PROCESS"
--   component instantiations:              "<ENTITY_>I_<#|FUNC>"
------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library proc_common_v2_00_a;
use proc_common_v2_00_a.proc_common_pkg.all;
------------------------------------------------------------------------------
-- Entity section
------------------------------------------------------------------------------
-- Definition of Generics:
--   C_DWIDTH                     -- User logic data bus width
--   C_NUM_CE                     -- User logic chip enable bus width
--   C_IP_INTR_NUM                -- User logic number of interrupt event
--   C_RDFIFO_DWIDTH              -- Data width of Read FIFO
--   C_RDFIFO_DEPTH               -- Depth of Read FIFO
--   C_WRFIFO_DWIDTH              -- Data width of Write FIFO
--   C_WRFIFO_DEPTH               -- Depth of Write FIFO
--
-- Definition of Ports:
--   Bus2IP_Clk                   -- Bus to IP clock
--   Bus2IP_Reset                 -- Bus to IP reset
--   IP2Bus_IntrEvent             -- IP to Bus interrupt event
--   Bus2IP_Data                  -- Bus to IP data bus for user logic
--   Bus2IP_BE                    -- Bus to IP byte enables for user logic
--   Bus2IP_RdCE                  -- Bus to IP read chip enable for user logic
--   Bus2IP_WrCE                  -- Bus to IP write chip enable for user logic
--   IP2Bus_Data                  -- IP to Bus data bus for user logic
--   IP2Bus_Ack                   -- IP to Bus acknowledgement
--   IP2Bus_Retry                 -- IP to Bus retry response
--   IP2Bus_Error                 -- IP to Bus error response
--   IP2Bus_ToutSup               -- IP to Bus timeout suppress
--   IP2RFIFO_WrReq               -- IP to RFIFO : IP write request
--   IP2RFIFO_Data                -- IP to RFIFO : IP write data
--   IP2RFIFO_WrMark              -- IP to RFIFO : mark beginning of packet being written
--   IP2RFIFO_WrRelease           -- IP to RFIFO : return RFIFO to normal FIFO operation
--   IP2RFIFO_WrRestore           -- IP to RFIFO : restore the RFIFO to the last packet mark
--   RFIFO2IP_WrAck               -- RFIFO to IP : RFIFO write acknowledge
--   RFIFO2IP_AlmostFull          -- RFIFO to IP : RFIFO almost full
--   RFIFO2IP_Full                -- RFIFO to IP : RFIFO full
--   RFIFO2IP_Vacancy             -- RFIFO to IP : RFIFO vacancy
--   IP2WFIFO_RdReq               -- IP to WFIFO : IP read request
--   IP2WFIFO_RdMark              -- IP to WFIFO : mark beginning of packet being read
--   IP2WFIFO_RdRelease           -- IP to WFIFO : Return WFIFO to normal FIFO operation
--   IP2WFIFO_RdRestore           -- IP to WFIFO : restore the WFIFO to the last packet mark
--   WFIFO2IP_Data                -- WFIFO to IP : WFIFO read data
--   WFIFO2IP_RdAck               -- WFIFO to IP : WFIFO read acknowledge
--   WFIFO2IP_AlmostEmpty         -- WFIFO to IP : WFIFO almost empty
--   WFIFO2IP_Empty               -- WFIFO to IP : WFIFO empty
--   WFIFO2IP_Occupancy           -- WFIFO to IP : WFIFO occupancy
------------------------------------------------------------------------------

entity user_logic is
  generic
  (
    -- Bus protocol parameters, do not add to or delete
    C_DWIDTH                       : integer              := 32;
    C_NUM_CE                       : integer              := 1;
    C_IP_INTR_NUM                  : integer              := 1;
    C_RDFIFO_DWIDTH                : integer              := 32;
    C_RDFIFO_DEPTH                 : integer              := 512;
    C_WRFIFO_DWIDTH                : integer              := 32;
    C_WRFIFO_DEPTH                 : integer              := 512
  );
  port
  (
     --USER ports added here
            FillSel: in std_logic;
            ShiftEn: in std_logic;
            Data_In: in std_logic_vector (15 downto 0);
            Data_Out: out std_logic_vector (15 downto 0);
    Bus2IP_Clk                    : in  std_logic;
    Bus2IP_Reset                  : in  std_logic;
    IP2Bus_IntrEvent              : out std_logic_vector(0 to C_IP_INTR_NUM-1);
    Bus2IP_Data                   : in  std_logic_vector(0 to C_DWIDTH-1);
    Bus2IP_BE                     : in  std_logic_vector(0 to C_DWIDTH/8-1);
    Bus2IP_RdCE                   : in  std_logic_vector(0 to C_NUM_CE-1);
    Bus2IP_WrCE                   : in  std_logic_vector(0 to C_NUM_CE-1);
    IP2Bus_Data                   : out std_logic_vector(0 to C_DWIDTH-1);
    IP2Bus_Ack                    : out std_logic;
    IP2Bus_Retry                  : out std_logic;
    IP2Bus_Error                  : out std_logic;
    IP2Bus_ToutSup                : out std_logic;
    IP2RFIFO_WrReq                : out std_logic;
    IP2RFIFO_Data                 : out std_logic_vector(0 to C_RDFIFO_DWIDTH-1);
    IP2RFIFO_WrMark               : out std_logic;
    IP2RFIFO_WrRelease            : out std_logic;
    IP2RFIFO_WrRestore            : out std_logic;
    RFIFO2IP_WrAck                : in  std_logic;
    RFIFO2IP_AlmostFull           : in  std_logic;
    RFIFO2IP_Full                 : in  std_logic;
    RFIFO2IP_Vacancy              : in  std_logic_vector(0 to log2(C_RDFIFO_DEPTH));
    IP2WFIFO_RdReq                : out std_logic;
    IP2WFIFO_RdMark               : out std_logic;
    IP2WFIFO_RdRelease            : out std_logic;
    IP2WFIFO_RdRestore            : out std_logic;
    WFIFO2IP_Data                 : in  std_logic_vector(0 to C_WRFIFO_DWIDTH-1);
    WFIFO2IP_RdAck                : in  std_logic;
    WFIFO2IP_AlmostEmpty          : in  std_logic;
    WFIFO2IP_Empty                : in  std_logic;
    WFIFO2IP_Occupancy            : in  std_logic_vector(0 to log2(C_WRFIFO_DEPTH))

  );
end entity user_logic;

------------------------------------------------------------------------------
-- Architecture section
------------------------------------------------------------------------------

architecture IMP of user_logic is


  ------------------------------------------
  -- Signals for user logic interrupt
  ------------------------------------------
  signal interrupt                : std_logic_vector(0 to C_IP_INTR_NUM-1);
  ------------------------------------------
  -- Signals for read/write fifo example
  ------------------------------------------
  type FIFO_CNTL_SM_TYPE is (IDLE, RD_REQ, WR_REQ);
  signal fifo_cntl_ns                  : FIFO_CNTL_SM_TYPE;
  signal fifo_cntl_cs                  : FIFO_CNTL_SM_TYPE;
  signal ip2wfifo_rdreq_cmb            : std_logic;
  signal ip2rfifo_wrreq_cmb            : std_logic;

component mb_simulator
Port (     Data_In : in std_logic_vector(15 downto 0);
           clk : in std_logic;
           ShiftEn : in std_logic;
           FillSel : in std_logic;
           Data_Out : out std_logic_vector(15 downto 0));
end component;

begin

mb_simulator_1: mbsimulator
port map ( Data_In, Bus2IP_Clk, ShiftEn, FillSel, Data_Out);

  ------------------------------------------
  -- generate user logic interrupts
  ------------------------------------------
  INTR_PROC : process( Bus2IP_Clk ) is
    constant COUNT_SIZE   : integer := 30;
    constant ALL_ONES     : std_logic_vector(0 to COUNT_SIZE-1) := (others => '1');
    variable counter      : std_logic_vector(0 to COUNT_SIZE-1);
  begin

    if ( Bus2IP_Clk'event and Bus2IP_Clk = '1' ) then
      if ( Bus2IP_Reset = '1' ) then
        counter := (others => '0');
        interrupt <= (others => '0');
      else
        counter := counter + 1;
        if ( counter = ALL_ONES ) then
          interrupt <= (others => '1');
        else
          interrupt <= (others => '0');
        end if;
      end if;
    end if;
  end process INTR_PROC;

  IP2Bus_IntrEvent <= interrupt;

  ------------------------------------------
  -- read/write fifo
   ------------------------------------------
  IP2RFIFO_WrMark    <= '0';
  IP2RFIFO_WrRelease <= '0';
  IP2RFIFO_WrRestore <= '0';
  IP2WFIFO_RdMark    <= '0';
  IP2WFIFO_RdRelease <= '0';
  IP2WFIFO_RdRestore <= '0';

  FIFO_CNTL_SM_COMB : process( WFIFO2IP_empty, WFIFO2IP_RdAck, RFIFO2IP_full,
RFIFO2IP_WrAck, fifo_cntl_cs ) is
  begin

    -- set defaults
    ip2wfifo_rdreq_cmb <= '0';
    ip2rfifo_wrreq_cmb <= '0';
    fifo_cntl_ns       <= fifo_cntl_cs;

    case fifo_cntl_cs is
      when IDLE =>
        -- data is available in the write fifo and there's space in the read fifo,
        -- so we can start transfering the data from write fifo to read fifo
        if ( WFIFO2IP_empty = '0' and RFIFO2IP_full = '0' ) then
          ip2wfifo_rdreq_cmb <= '1';
          fifo_cntl_ns       <= RD_REQ;
        end if;
      when RD_REQ =>
        -- data has been read from the write fifo,
        -- so we can write it to the read fifo
        if ( WFIFO2IP_RdAck = '1' ) then
          ip2rfifo_wrreq_cmb <= '1';
          fifo_cntl_ns       <= WR_REQ;
        end if;
      when WR_REQ =>
        -- data has been written to the read fifo,
        -- so data transfer is done
        if ( RFIFO2IP_WrAck = '1' ) then
          fifo_cntl_ns <= IDLE;
        end if;
      when others =>
        fifo_cntl_ns <= IDLE;
    end case;

  end process FIFO_CNTL_SM_COMB;

  FIFO_CNTL_SM_SEQ : process( Bus2IP_Clk ) is
  begin

    if ( Bus2IP_Clk'event and Bus2IP_Clk = '1' ) then
      if ( Bus2IP_Reset = '1' ) then
        IP2WFIFO_RdReq <= '0';
        IP2RFIFO_WrReq <= '0';
        fifo_cntl_cs   <= IDLE;
      else
        IP2WFIFO_RdReq <= ip2wfifo_rdreq_cmb;
        IP2RFIFO_WrReq <= ip2rfifo_wrreq_cmb;
        fifo_cntl_cs   <= fifo_cntl_ns;
      end if;
    end if;

  end process FIFO_CNTL_SM_SEQ;

  IP2RFIFO_Data <= WFIFO2IP_Data;

  ------------------------------------------
  -- drive IP to Bus signals
  ------------------------------------------
  IP2Bus_Data        <= (others => '0');

  IP2Bus_Ack         <= Bus2IP_WrCE(0) or Bus2IP_RdCE(0);
  IP2Bus_Error       <= '0';
  IP2Bus_Retry       <= '0';
  IP2Bus_ToutSup     <= '0';

end IMP;
```

VITA

Wei Tang

Wei Tang was born in Hunan Province, the People's Republic of China in 1968. He received his Bachelor of Science in Precision Instruments and Mechanology department from Tsinghua University in July 1991. In 1998, He graduated from Chinese Academy of Space technology (CAST) with his Master of Science in Computer Engineering. After that, he came to the United States of American to purse a Ph.D. degree in Industrial Engineering. During the first year, he studied in the University of Toledo, Toledo, OH. In 1999, 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 from Virginia Polytechnic Institute and State University in December 2005. Currently he is seeking a position in manufacturing industry.