DESIGN AND EVALUATION METHODOLOGY FOR COMPUTER-CONTROLLED MANUFACTURING SYSTEMS

by

Harold A. Scott Jr.

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 Engineering and Operations Research

APPROVED:

R. P. Davis, Chairman

M. H. Agee

J. M. A. Tanchoco

C. E. Nunnally

R. A. Wysk

December 1982

Blacksburg, Virginia

ACKNOWLEDGMENTS

A special thanks to Dr. Robert P. Davis for chairing the committee and helping to bring this research in on time and under budget.

Thanks to the members of the committee, Drs. Richard A. Wysk, Charles E. Nunnally, Jose M. A. Tanchoco, and Marvin H. Agee, for their time and support.

Also thanks to of General Electric Company for his support of this research.

TABLE OF CONTENTS

TABLE OF CONTENTS (CONTINUED)

TABLE OF CONTENTS (CONTINUED)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (CONTINUED)

LIST OF ILLUSTRATIONS (CONTINUED)

LIST OF TABLES

CHAPTER ONE

INTRODUCTION

Computer control is being implemented at all manufacturing system levels to increase productivity and flexibility. Computer controllers range in size from small microprocessors directing machining operations to mainframes supervising an entire manufacturing facility. The ultimate goal of computer control implementation is the computer-integrated manufacturing (CIM) system, which combines computer controllers at each manufacturing level into a single control network. The effectiveness of computer-controlled manufacturing systems is directly related to control strategy and hardware design.

A major problem in CIM implementation is the complexity of designing and evaluating control hardware and software at all levels of the manufacturing operation. Many different types of hardware control configurations are commercially available for controlling a given process. Evaluation and design of these control systems require a full working knowledge of the process being controlled and computer control technology. The most important and often overlooked consideration in design and evaluation of a computer-controlled manufacturing system is the actual process being controlled. Therefore, computer control systems must be evaluated in terms of the manufacturing process, as well as controller cost and performance.

Research Objectives

The purpose of this research is to develop a comprehensive design and evaluation methodology for computer-controlled manufacturing systems. This methodology can be used to evaluate all levels of the manufacturing process, from adaptive control of machining operations to the hierarchical control network of an automated factory. A two-step approach is employed in the methodology. First, a mathematical model is used to identify a cost-effective, feasible design. Basic control hardware performance characteristics are determined by manufacturing system control requirements. Once hardware components are identified, a simulation model is used to evaluate performance characteristics of the control system with respect to the manufacturing process. Simulation modeling determines the feasibility of control system dynamics. Actual components for the control system can be specified from simulation model results. A prototype or actual system can then be implemented from methodology results.

Control System Evaluation Methodology

The manufacturing control system evaluation methodology developed in this research is outlined in Figure 1.1. The two types of analysis employed by the methodology are: (1) mathematical modeling and optimization, and (2) simulation modeling and experimentation.

The mathematical model is a screening process to analyze basic designs and components, given process control requirements and component costs. Many control strategies and designs can be evaluated at this stage. Technological characteristics of components make the

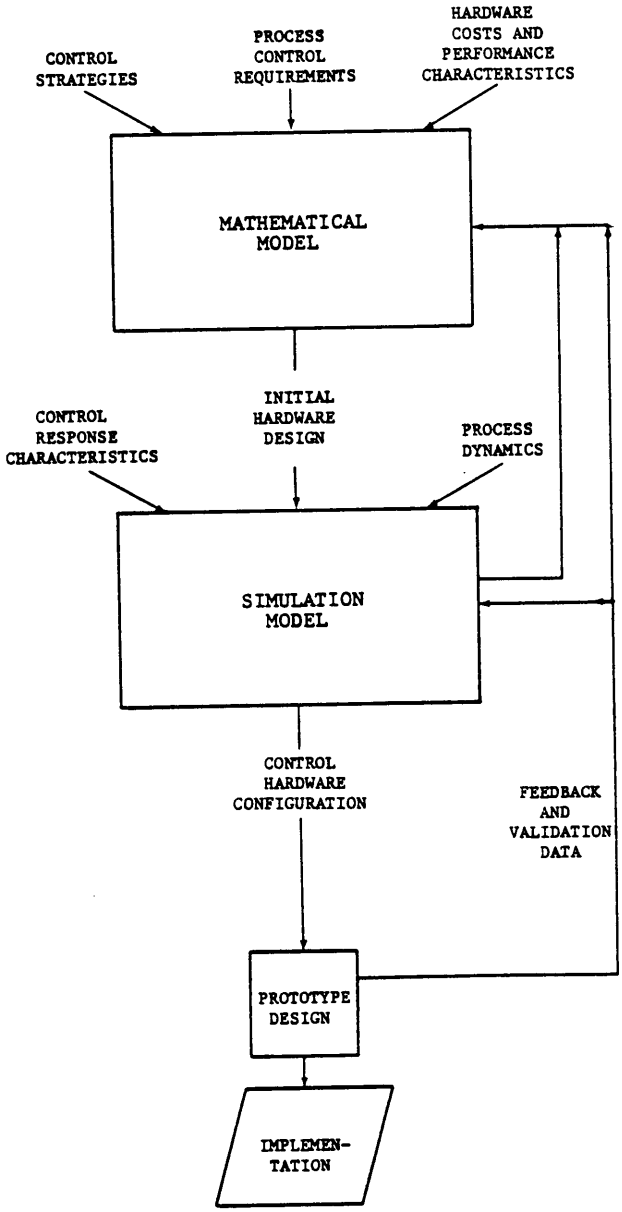

Figure 1.1. Control System Design and Evaluation Methodology

solution procedures quite simple. The most cost-effective, feasible design configuration is further evaluated with a simulation model. Dynamic, time-dependent system response characteristics are analyzed by the simulation model. Simulation analysis requires more detailed data on the control system identified by the mathematical model. Simulation model development time is substantially reduced, because the mathematical model is used to identify the initial system design. Actual components can be specified and evaluated during simulation. Simulation results can also be used with postoptimality analysis of the mathematical model to evaluate components which do not meet specified control requirements. For instance, if a computer does not have sufficient storage capacity, the mathematical model postoptimality analysis can directly identify the cost and storage unit to meet the storage requirements without re-solving the mathematical model. The methodology can be applied to all levels of the manufacturing process in varying degrees of detail.

The evaluation and design methodology is dynamic. Evaluation and component selections are based on manufacturing process control requirements. Thus, the methodology is able to evaluate most computer-controlled manufacturing systems. Feedback data provides additional information. As more evaluations are performed, the methodology data base becomes more accurate.

Once the process control system has been simulated, a prototype can be built. All process control system components are identified by the mathematical and simulation model analyses. If the simulation

model is very detailed in describing the control system and process, a system can be implemented directly from model results. Complex systems with high process variations and many control variables will probably require that a prototype or working model be constructed and analyzed. Results from the prototype analysis can be used to validate and update the simulation model data base. The final design is obtained from the prototype analysis, and the control system can be implemented. Information can be used from the prototype and process design to further validate both the mathematical and simulation models, providing additional system operational data.

Mathematical Model

Since many types of computer control components and designs are usually under consideration, a procedure must be developed to determine a cost-effective hardware configuration which can satisfy the process control requirements. For example, an adaptive control system maximum response time for a machining process is 23 microseconds; that is, the control system requires 23 microseconds to detect the machining process variables, process the sensor signals, make a control decision, and send the decision to the machine control servo. As this example illustrates, control hardware and software component characteristics must be selected to satisfy process control requirements. In addition, software, hardware, implementation, and operating costs must be considered in the selection of a component configuration.

A computer control system can be expressed as a resource allocation model. The component performance characteristics are the

resources allocated to satisfy process control requirements. All control requirements must be satisfied; otherwise, the control system is infeasible. Performance requirements are defined by the process being controlled. By having the process dictate control requirements, control system designs are evaluated with respect to the process.

Control system components can be defined in terms of performance characteristics (e.g., response time, resolution, and processor speed). Since components are defined in terms of basic performance characteristics, the model is not technology dependent. Different kinds of technology can be evaluated simultaneously by describing components in terms of basic model parameters. New technologies can be analyzed directly with existing technologies.

Component cost can be expressed as a function of performance characteristics. Thus, an optimal minimum-cost solution can be obtained from the resource allocation model, subject to process control design requirements. The mathematical model solution identifies all hardware components in the minimum-cost configuration. With an appropriate solution procedure, postoptimality analysis can be performed. With this analysis, different components can be examined relative to the optimal solution. Reliability analysis can also be performed to evaluate system reliability relative to component cost and feasibility.

The mathematical model identifies a basic relationship between component costs and operational constraints. This relationship enables system cost to be analyzed during the initial design phase of both the

manufacturing and computer control systems. The mathematical model is the critical link between the manufacturing process and computer control system design. By expressing manufacturing process requirements in terms of the mathematical model, the manufacturing system designer can communicate directly with the designer of the computer control system.

Simulation Model

Once an optimal control system is identified by the mathematical model analysis, the dynamic behavior of control system hardware and software must be investigated. While the mathematical model defines system components on the basis of their performance characteristics, process dynamics and control strategies must be evaluated before actual implementation. A digital computer simulation model is used to perform this analysis. A computer control system simulator provides the designer with the data required to determine individual component and total system performance measures.

Two simulation models are combined to evaluate an entire system. First, a simulation model is developed for the process being controlled by the computer control system. Process variability is simulated and related directly to control actions. These control actions correspond to the process control variables identified in the basic control strategy. The process simulation is the most critical and important. Second, the computer control system is simulated. All dynamic responses and delays are included for interface and communication equipment. In addition, control software logic is integrated into the

simulation model. Computer control system components are simulated in a modular structure to enable other components to be added and evaluated without changing the original model's structure. The simulation model can analyze many levels of computer control. Variations in the computer control network are related directly to process variations.

A computer control system is responsive to process variations. These variations can be both continuous or discrete over time. For example, the torque of a motor controlling spindle speed changes continuously over time. A control signal from a minicomputer is a discrete event. Thus, the simulator has to execute both discrete and continuous simulation simultaneously. Since this methodology has to evaluate all manufacturing levels, the simulator must also be flexible.

The simulation model is used to evaluate the following major component types and several operating characteristics of each type.

1. Computers and processors: interrupt input/output (I/O), real-time control logic program execution
2. Communications: duplexing/wideband, message protocol, local area networks
3. Process control interfaces: real-time control and delays, sensor/data acquisition, analog processes and devices

Unlike the mathematical model, which describes all the components and computers by deterministic performance characteristics, the simulation model must capture dynamic, probabilistic, real-time operating characteristics.

Different hardware performing the same function might have completely different operating procedures. For instance, a midlevel control computer could use either an interrupt-driven or a pooled system for servicing machine requests. Simulation of the interrupt system would be completely different from pooling routine simulation. The simulator must be extremely flexible to include all components under consideration. This flexibility is extremely important if sensitivity analysis is to be performed with the simulation model.

Finally, simulation output must be in a format that can be used to evaluate control system designs. Means and utilizations of controllers are important. However, the simulation must trace the entire control action over the entire control time domain. Responses and control actions have to be identified and recorded at precise time intervals. Process states have to be identified at specific times. Simulation model output must include memory requirements and software execution times. Processor failures or saturated communication channels must also be documented. The events leading up to a failure must give a clear picture of the cause of the failure. Without properly simulated variables and control states, the simulation model will not be an effective design tool. The more detailed the simulation model, the better the understanding of the process and control relationship.

Research Procedures

Chapter 2 reviews the evolution of computer control systems for automated manufacturing processes. Background information on application of the mathematical and simulation models to automated systems is also provided. Chapter 3 develops a hierarchical control system mathematical model and an analysis procedure using dynamic programming solution techniques. Chapter 4 presents a mathematical model for an adaptive control system and a general procedure for selecting control hardware components, based on system performance requirements. Solutions are given for both models, and examples illustrate the solution procedures. Postoptimality analysis techniques are discussed for the mathematical models.

Chapter 5 covers simulation model development. The Simulation Language for Alternative Modeling (SLAM) is used for the simulation analysis. An example manufacturing cell is simulated, illustrating the versatility of the analytical procedure. Results and conclusions of the research are presented in Chapter 6. The extension of the methodology to evaluate other manufacturing systems, such as robotics, is also discussed.

Appendix A provides the source listing of the example manufacturing cell SLAM program discussed in Chapter 5. Appendix B describes simulation of a computer numerically controlled (CNC) turning process. The adaptive control method is analyzed and illustrates the continuous simulation capability of SLAM. In addition, graphic output capabilities developed during this research are demonstrated.

Development of an advanced optimal adaptive control procedure with compensatory tracking is discussed in Appendix C. This method dynamically controls cutting speed, based on optimal tool life and flank wear characteristics. The adaptive control program is designed to take advantage of microprocessor architecture. This application illustrates the model's capability to analyze complex continuous systems.

CHAPTER TWO

LITERATURE REVIEW

Fully automated production systems are possible with today's computer and microelectronic technology. These systems have substantial impact for the batch-type machine shop, where only five percent of average part production time is spent on a machine [46]. A computer-controlled manufacturing system can reduce the amount of idle time by dynamically scheduling and controlling machine, transfer, and storage operations throughout the entire production process.

Fully automated production systems are completely computer-controlled. Computers make all the decisions concerning production schedules and processes. Computers also control numerical control (NC) machines and transport mechanisms located in the plant. The computers ensure optimum machine utilization and throughput service times for variable mixed part processing. Once demand is specified and part priority established, computers monitor and control the entire production process, from selecting a raw resource to storing finished products in warehouses.

This chapter reviews the literature on the evolution of computer-controlled manufacturing systems over the past thirty years. Current analysis procedures for developing control computer configurations are reviewed. Present mathematical modeling and simulation techniques for manufacturing computer control systems are also examined.

Development of Computer Control

Implementation of computer control at all levels of manufacturing has followed the rapid growth of the computer and microelectronics industry. The NC system developed by the Massachusetts Institute of Technology in 1952 [77] paved the way for computer control of manufacturing processes. NC machines are directly compatible with the digital computer. International Business Machines Corporation introduced the System/360 in 1965 [11]. Two years later, both the direct numerically controlled (DNC) and computer numerically controlled (CNC) machines were developed. Sunstrand Corporation was able to control 256 Omnimils with one IBM System/360 Model 50 [24]. This initiated the "star" network control phase of manufacturing processes.

Computers had to be housed in a controlled environment away from the manufacturing processes. Mainframes, such as the IBM System/360, cost millions of dollars. In addition, connecting all the machine control functions was very expensive, and the network of wires was complex to maintain. With the high cost of computer control, a computer's full capability had to be used in order to justify the system [63]. Thus, control of the manufacturing process was centralized. All control loops to DNC machines and other processes were connected to a large computer in a central location. All control programs were executed by the central computer. Manufacturing system reliability was totally dependent on the central computer. During the late 1960s, DNC machines could receive and process only one or two part program instructions at a time [57]. The main computer had to

constantly "pool" and send instructions to the DNC machines. If the main computer went down, so did the manufacturing process under the computer's control.

The hard-wired controllers of the DNC machines also limited manufacturing flexibility. In 1972, Sunstrand Corporation introduced the soft-wired integrated numerical controller (SWINC) system, which enabled machines to handle different types of part operations by changing the machine control program [78]. In the same year, Intel developed the complementary metal-oxide semiconductor (CMOS) microprocessor, which began the drive to implement microelectronic controls at the process level [30]. Two years later, microprocessors were being used in machine tools [78].

The automated factory moved closer to reality when Digital Equipment Corporation (DEC) introduced a distributed plant management system for controlling remote stations in 1977 [78]. Microprocessors were "hardened" so that microprocessor controllers could be placed at the process level. Machines, material transfer devices, control computers, and microprocessors no longer had to be in constant communication with a centralized computer. With these developments, control of the manufacturing system started to become distributed. Control decisions were closer to the manufacturing process, increasing manufacturing system reliability and decreasing communication requirements.

In the mid-1970s, group technology was being considered in the United States [78]. The group technology concept was developed by

S. P. Mitrofanov of the U.S.S.R. in 1946 [22]. Germany and the U.S.S.R have employed group technology in their manufacturing processes since 1950. In this country, the reason for the switch to group technology from transfer lines was the fact that 60 to 70 percent of the discrete parts manufactured in the U.S. are produced in small lot sizes of 50 or less [19, 60]. Parts spent an average 95 percent of the total process time either being transferred or waiting for processing; actual part machining time was only 30 percent of the five percent remaining for machining setup and processing [46]. With group technology, machines are grouped together in a manufacturing cell with high flexibility for manufacturing the many different small part lots, increasing productivity and machine utilization.

Group technology was the main design concept of the first complete variable mission manufacturing (VMM) cell developed by Cincinnati Milacron in 1980 [78]. The VMM cell comprised a group of machining centers controlled by a minicomputer with automatic transfer mechanisms between all the centers. The entire system is programmed for different batch machining operation requirements. Machining operation sequences can be changed in seconds for different part batches.

In order to control the cells and individual machines, the Modway Public Industrial local area network was developed by Modicon Division of Gould Incorporated in 1978 [2]. This "data highway" enabled processors to address devices outside the physical computer, as well as allowing communication between micro- and mini- control computers of different vendors. The data highway communication network links CNC

machines, DNC machines, machine cells, transfer and storage systems together, making a CIM system feasible. The distributed control network is the future trend in computer control systems, as controllers and digital control devices are implemented closer to the process [32, 33, 37, 42]. For instance, advanced robotic systems assign a microprocessor to control movements of each axis [20].

Implementation of Computer Control in Manufacturing

Computer control implementation has been successful at all levels of manufacturing. A Bendix optimizing adaptive control machining system built for the U.S. Air Force increased milling and turning operations 20 percent over conventional methods [79]. At the cell level, the IBM Quick Turn Around Time (QTAT) manufacturing line was developed to speed fabrication of customized logic circuits. IBM computers, including multiple System/370 Model 168s, control over one hundred automated tool subsystems grouped into eight sectors, making the entire custom chip from a master slice. The entire IBM process is computer controlled from design to final inspection. The IBM QTAT system substantially shortens production cycle time [41]. Such systems have been integrated into a computer-aided design (CAD) system and other computer-controlled manufacturing processes, providing significant reductions in logic chip manufacturing costs. As these systems illustrate, large increases in productivity can be realized by implementation of computer control in manufacturing.

With successes such as the IBM QTAT system, computer control of the manufacturing process seems to be the answer to many manufacturing problems. As one article recently stated, "the best way for a plant to tighten its belt is with a computer" [56]. However, there have also been many computer control implementation disasters. Documentation on computer control system failures is scarce, since most companies choose not to publicize mistakes. Investigations into these failures indicate the manufacturing system was the cause of the poor performance, because it was inadequately designed for the computer control system [56].

Manufacturing management has been "throwing computers in the general direction of problems, without any clear idea of exactly how the computer is supposed to solve the problem" [56]. The trend has been for the manufacturing engineers to let the computer designers develop the entire control system without a well-defined understanding of the manufacturing process being controlled [34]. In 1980 C. R. Williams of Sperry Univac noted that "the responsibility for implementing computer-based manufacturing systems in most companies is in the wrong hands," referring to the computer professionals who usually perform most of the design work [73].

Most current computer-controlled manufacturing systems comprise a computer or minicomputer connected to a group of NC machines. The computer has specific programs for each machine processor. Machine requirements are based on scheduling and routing techniques most frequently obtained from predicted demand and resource availability. The workpiece is routed through the system and waits at a busy or

failed machine. There are other machines in the system which can perform the same operation, but the workpiece is not routed to another machine because the computer does not have enough memory to support two NC machines with the same operations program. This example illustrates current computer control of machines, which at best has achieved only 55 to 68 percent machine utilization [40].

The main failure in this type of control system is inflexibility once the workpiece is routed into the system. Real-time control is not present in such a system. If a new demand or priority arises, the system has to completely discard its current operation. This is not the type of computer-controlled system that can efficiently operate a manufacturing process, much less an entire manufacturing system.

Two observations have been made concerning the problems of controlling manufacturing systems [72]: (1) resources should be allocated to tasks as close to their initiation as possible [50], based on the most recent information, and (2) machines should be pooled so that one machine can perform another's operation. The system must be flexible with the shortest possible dwell time for each machining operation. The system must be interconnected with communications so command and control messages can be received and acknowledged to ensure control information is not deleted from the system. Constant system supervision must be maintained to detect failures and monitor impending job completions. The computers should control the entire system throughput and utilize all machines by means of a transportation network which can bypass failed or busy machines.

Design and Evaluation of Computer-Controlled Manufacturing

Design and evaluation of a computer-controlled manufacturing system comprises a large percentage of the total implemention time. On a small CIM system with just a few manufacturing cells, total implementation time could be as long as five years, with costs approaching fifty million dollars [64]. More than half that time is spent evaluating and designing system control hardware and strategies. In the typical computer control implementation, up to 95 percent of total implementation time is spent on design and evaluation of the control and manufacturing system interaction [69]. In a manufacturing control system, many different control levels must be examined; response characteristics and interactions must be related to the entire process control system, beginning with the process being controlled.

Manufacturing and Computer Control Systems Analysis

Design and evaluation of a manufacturing control system is presently performed by a committee of engineers, machine tool builders, and computer systems analysts. There are some guidelines for starting the design of the system [64]. However, the system design phase is usually divided into two areas: the manufacturing process and computer control system. Design and evaluation techniques have to be established for each area. Thus, process designers specify schedules and equipment required for the manufacturing system, while computer software and hardware design engineers design the control system.

Two basic techniques are used to evaluate the production process and computer control system before a prototype or actual system is

built: mathematical modeling and simulation modeling. Although mathematical and simulation models are used to evaluate both the production process and computer control system, the actual evaluation procedures have been developed separately.

Mathematical models are used to evaluate the levels of computer control in manufacturing systems. Linear programming and dynamic programming models have been developed for determining assembly line balancing and evaluating machine idle time, based on cycle time and shortest path through a network. Integer programming and dynamic programming are deterministic modeling techniques and have been applied primarily to sequencing, network capacity, inventory, and material requirements problems.

Queuing models have been developed to analyze flow and capacity characteristics. Models, such as the Computer Analysis of Networks of Queues (CAN-Q) [72], evaluate the flow and capacity of cellular manufacturing systems. Another stochastic model is the Generalized Network Simulator (GNS) [43], which is used to analyze production systems with in-process inventory and parallel machine stations. Stochastic models are also designed to investigate characteristics of the individual cell [10]. These stochastic models deal with system loading, part flow, and capacity.

Computer control networks are evaluated in a similar manner. Queuing models are used to determine system capacities over a given time domain [9]. A dataway network queuing model has been developed for evaluating communication traffic demand from manufacturing

operations [5]. Like manufacturing stochastic models, computer network models are used for capacity planning, system utilization, and reliability analysis.

Simulation

Simulation models are used to evaluate CIM designs. Both General Motors and Caterpillar Tractor have used general simulation languages to design and evaluate new flexible manufacturing cells [7]. These evaluations included machine utilization, scheduling, and part process rates as performance variables. A few of the many examples of discrete-event simulations of automated manufacturing systems are found in references [45, 59, 66]. All these examples assumed that the computer control systems were process-oriented; distributions were assumed for machine and transfer cycle times. Manufacturing capacity analysis was the main objective of these applications.

Many discrete-event simulators have been developed for evaluating flexible manufacturing system performance [1]. These include IIT Research Institute's Manufacturing Simulator and Multi-Routine Simulator, Cincinnati Milacron's VMM General Simulator, and Purdue University's General Computerized Manufacturing Simulator (GCMS) [7]. These simulators are used to support optimal planning of CIM systems. Other simulators have been developed for general production facility simulation [6, 48, 55, 61]. All performance variables in these simulators deal with the manufacturing system, machine and transfer utilization, system capacity, resource planning requirements, and inventory control.

The computer system simulation models usually estimate computer utilization based on program-generated loads [65]. Statistical or mathematical models are imbedded in the simulation model to approximate computer performance characteristics, based on system states [8]. Even hybrid regression is employed to model the computer because of the often prohibitive cost of simulation time [17]. Computer capacity and reliability are usually evaluated with these simulation techniques [51, 52].

Few simulation applications have been developed to evaluate software [14, 25]. Software is analyzed with respect to capacity. Software programs are generated as discrete events with variable service times sampled from known distributions.

Emulator programs are the only simulation models which can evaluate software and control actions in a real-time context [67]. These emulators run software with actual hardware. Usually several processors can be emulated with the same emulation software; circuits are developed with a CAD system [62]. Distributed network control software has been developed using a data communications computer emulator network tied into an actual physical system.

Network computer simulation models have also been developed to analyze system capacity. Task loads are generated into the system and various devices in the network are simulated [21]. The data highways used in manufacturing control are also analyzed with simulation models [5]. Again, these simulation models are general in nature and are

primarily used to analyze network message traffic. The computers and devices generate events which contend for use on the data highway.

Finally, process dynamics have been modeled using differential energy conservation equations, analog computers, and digital simulations [23]. Differential equations have been used to describe machine tool metal removal dynamics [35, 36] over a continuous time domain. These differential equations can be extremely accurate in describing system dynamics [68]. From these equations, system responses to inputs are used to design control systems. Digital simulation software packages, such as the Advanced Continuous Simulation Language (ACSL) [47], have been developed to simulate complex analog control systems over continuous time domains [32]. These digital simulation programs are used directly in manufacturing process control system design. Except in advanced digital simulation, random variability is not included in the models [23].

Summary

Each level of the manufacturing control system and process has been analyzed using various mathematical and simulation modeling techniques. Models with their current capabilities and formats cannot be used directly in an optimal design process to select system components for each level of the manufacturing control system based on cost and performance requirements. In addition, process dynamics cannot be analyzed with control hardware and software in a real-time simulation.

At present, manufacturing process control analysis techniques have not been fully integrated with computer system design. A comprehensive design and evaluation methodology is developed in the following chapters. The methodology enables all parties in the CIM system design process to analyze each control component level, based on optimal cost design and system dynamics.

CHAPTER THREE

HIERARCHICAL CONTROL MATHEMATICAL MODEL

The trend in recent years has been to control automated production systems from raw materials to finished products entirely by computer. Hierarchical computer systems have proved the most effective method of production system control because of their modularity, reliability, and control functions [64]. A major problem in implementing a hierarchical control system (HCS) is identifying the most cost-effective computer configuration to control an automated manufacturing system. A method must be developed to determine the optimal number and type of computers and communication links for each level of the system to result in the lowest control costs.

In this chapter, a general mathematical model is formulated to describe HCS components in terms of computer/communication speed and capacity. The model relates production process requirements directly to computer control requirements. Once production requirements are established and processes are defined, dynamic programming techniques can be used to determine a minimum-cost hierarchical configuration as a function of cost and available computer resources. The model can be applied to any HCS configuration and is not dependent on current technology. By minimizing costs, the model identifies an optimal control system for the specified production process.

Basic Hierarchical Control System Levels

The basic control system for NC machines and automated material handling devices has three major functions: strategic control, throughput control, and process or tactical control. Figure 3.1 illustrates an HCS. At the top of the system is the strategic computer, a mainframe which handles most of the planning, forecasting, scheduling, and controlling functions of the plant or entire company. Computer-aided design (CAD) is supported at this level. This computer also performs many of the "housekeeping" tasks, such as billing, payroll, and recordkeeping. Its output relative to actual production processes is very general, e.g., the number of different products required, production goals, and future requirements. The strategic computer contains and determines the broad and general production requirements for plant processes. Its data base holds the information needed by managers to determine sales and predict demand.

General demand and production level output from the strategic computer must be translated into instructions for controlling the production process. The throughput control computer performs this task and controls the NC systems. It is at the center of the production control system. The computer determines which machining operations are required to meet demand, the resources available for each operation, and the NC machines required to perform the work. It constantly monitors feedback from different processes in the system and adjusts for changes in system status. Its memory stores data on materials and parts available and finished goods. Computers maintain master control

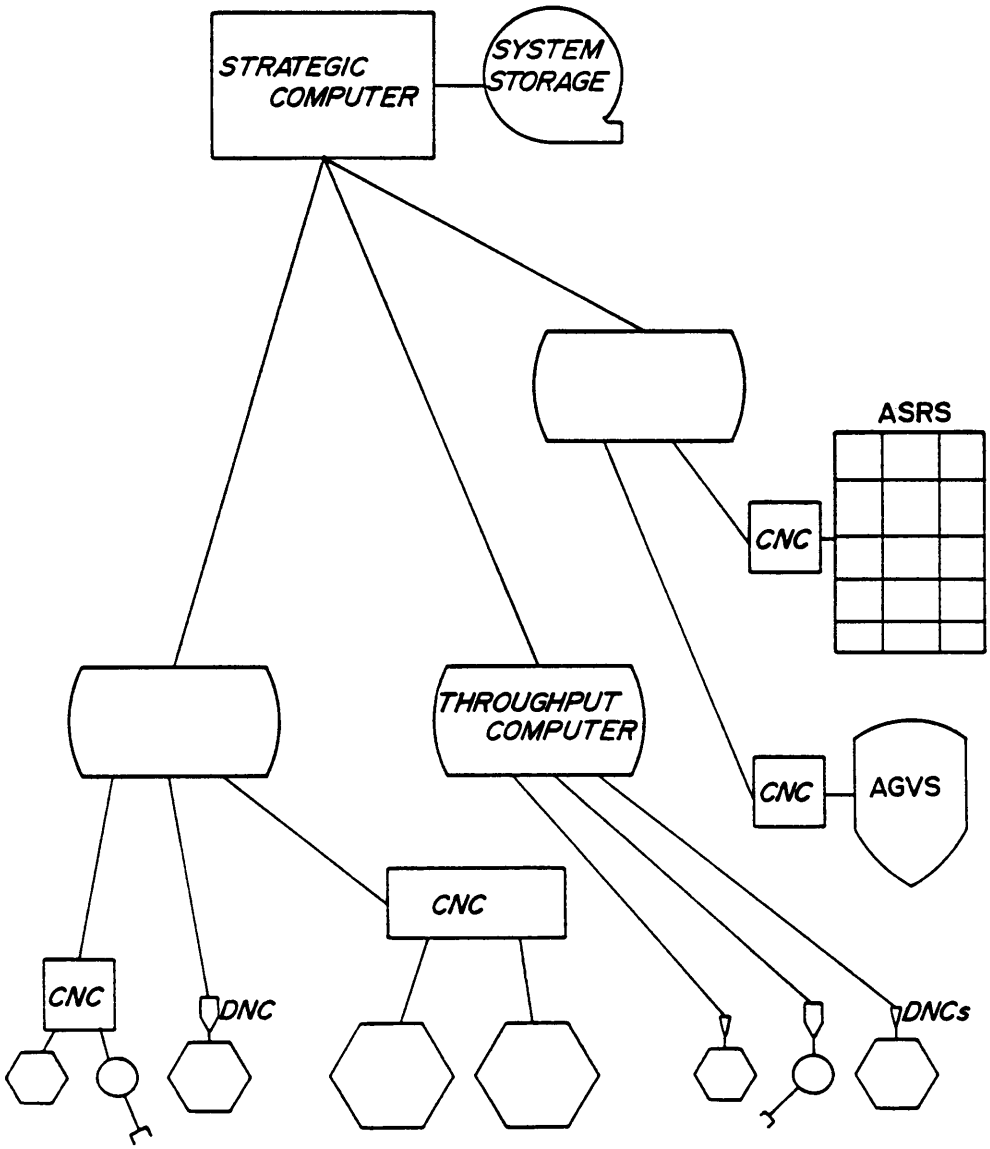

Figure 3.1. Hierarchical Control System

and status of the production process. If the computer's offline and online storage is large enough, the throughput computer can keep track of each part's location in the production process.

Tactical microprocessors or minicomputers control the NC machines and the process transportation mechanisms. These tactical controllers can have two-buffer direct numerical control (DNC) with a behind-the-tape reader (BTR) or complete computer numerical control (CNC) with all part programs stored in memory. The type of tactical control determines the processing load and communications the controlling throughput computer must have to ensure the machine's process functions can be performed. A DNC machine with BTR would require constant control during processing, while a CNC machine would require only a single process operation message. Tactical controllers for DNC machines are usually the least expensive of the NC machine control options. However, computer control of the DNC machine could make it the most expensive NC machine control option.

These control functions are the basis of an HCS for a production process. HCS levels are aggregated as a function of their direct control over the manufacturing process. The strategic computer deals with general information on production requirements and requires only general statistical data on the production process. The throughput computer interprets the requirements from the strategic computer and transforms them into the actual process required. The tactical computer sends the instructions required by tactical controllers to perform the required process operations. The computer monitors the

process resources and updates the status of every machine process. The throughput computer directly controls the tactical computers for the current operation. Statistical process data is relayed back to the strategic computer by the throughput computer.

A computer control system can be effectively modeled based on the following concepts. The model must represent the entire production process, including all communication and computer operations. An HCS must be capable of responding to variations in production operations. The most important aspect of the model is relating system control costs to manufacturing process requirements. An HCS can be viewed as a resource allocation procedure. The process creates a demand for control, and the HCS meets the demand.

Hierarchical Control Manufacturing System

Development of an HCS model should start at the process being controlled. Solberg, Barash, and others have simulated and developed manufacturing process configurations for hierarchical control [11, 28, 40, 72]. Pooling (or group technology) was identified as the best method of processing by computerized manufacturing systems [40]. By having NC machines programmed to do many basic jobs, flow through the production process can be greatly improved. The smaller programs can decrease part dwell time at the process station. The main strategy is optimizing machine utilization and providing system flexibility. Inflexibility is the main shortcoming of current computer-controlled manufacturing systems. The computer HCS must be able to respond instantly to changes in demand and resource requirements.

Workpiece transportation and process facility layout are the keys to process system capability. In a computer-controlled manufacturing system, the workpiece must be able to bypass any work station. This bypass capability substantially increases machine utilization and workpiece flow in simulations of data flow computer-operated manufacturing systems (CMS/DF) developed at Purdue University [40]. The bypass capability allows the system to continue processing with partial system failures. The main transportation system must connect all NC machines in the system. Figure 3.2 shows the basic layout of a computer-controlled system.

Some automated manufacturing process systems do not have buffers at the NC machines, eliminating in-process part storage completely. Without in-process storage, part transfer delays can decrease machine utilization. On the other hand, large part storage at work stations will degrade total system flexibility by increasing dwell time. To resolve this problem, an in-process buffer at each work station temporarily holds the part to be processed next. The in-process buffer increases machine utilization and process throughput [10].

The most important criterion in facility layout is the capability to bypass any machine by any workpiece. The system must be able to decouple the manufacturing process. A decoupled system greatly increases control system flexibility.

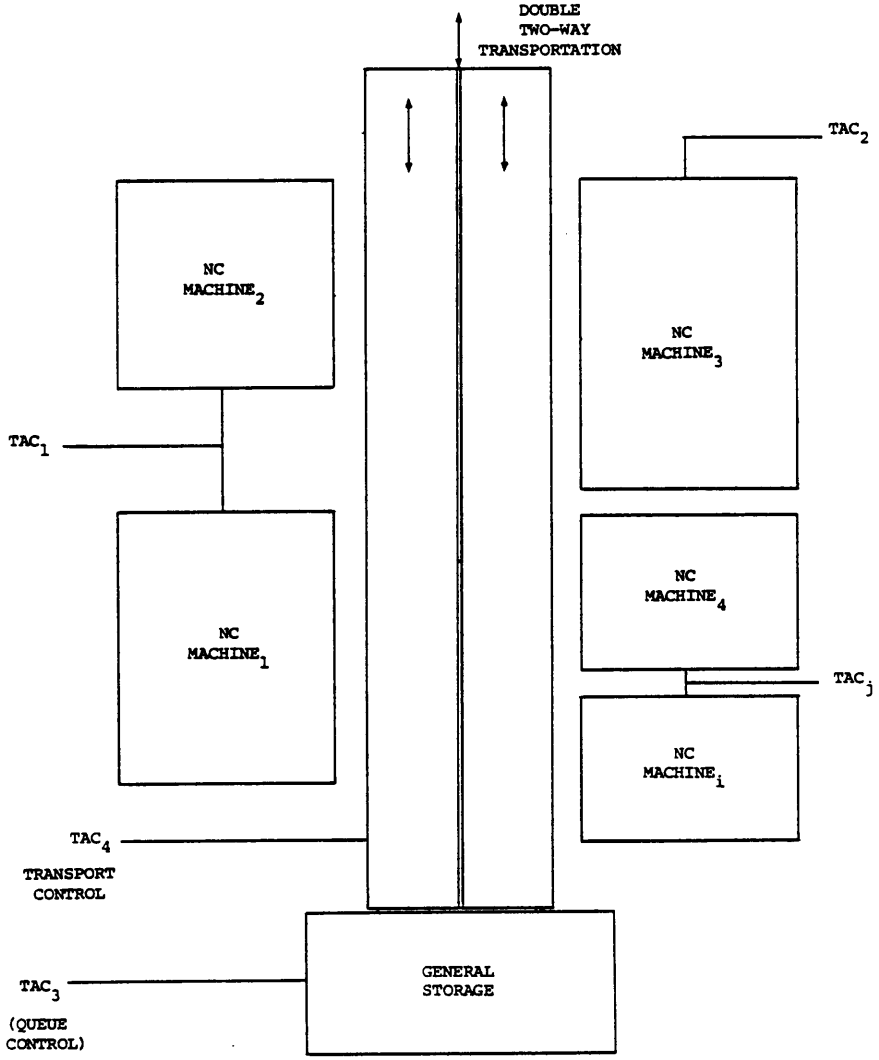

Figure 3.2. Hierarchical Control Manufacturing Facility Layout

Hierarchical Control Variables and Costs

Basic HCS components (Figure 3.3) are a central processing unit (CPU), processing memory, storage, I/O buffers, and communication links. The entire HCS can be described in terms of its component operations. Basic costs of a computer-controlled system are associated with these components. The HCS can be modeled using basic component costs. By minimizing costs of the control system model within these constraints, an optimal control system can be identified. The model can specify the basic requirements for computer and communication network equipment. The model's constraints can be changed to reflect advances in microelectronic technology.

The HCS model is developed by first describing each control level in terms of the basic components. Constraints are determined for each component. All control levels are then related as functions of basic component costs and the constraints. The objective function is determined from the cost relationships.

Basic System Components and Cost Relationships

Every computer has three basic systems: the CPU, memory, and I/O [38]. The CPU performs all the basic functions, such as addition and subtraction. The CPU executes the decision processes according to program instructions. The CPU is usually described by the number of bytes per word, registers, and execution speed. Processor speed is considered in this model. The process memory contains the data and programs required by the CPU to perform computations. The results of

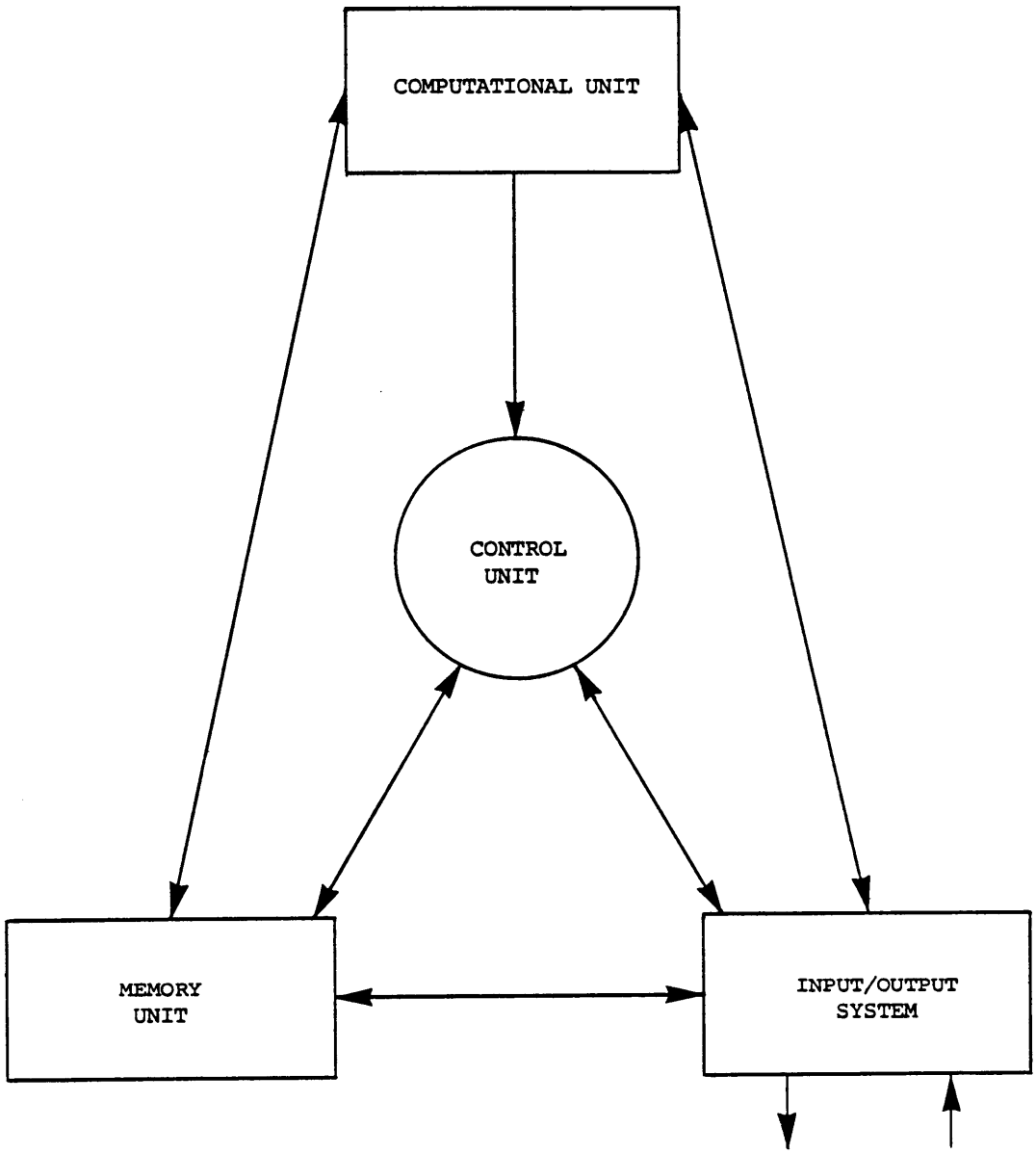

Figure 3.3. Computer Operating System Basic Components

the computations can also be stored in memory. In addition, data from the I/O can be stored directly into the memory for execution by the CPU. The memory storage size and retrieval speed are the limiting factors on most computer systems [54]. Both size and speed increase the cost.

The I/O ports are connected to both the memory and CPU. Each device is connected to a central controller. The central controller is a scheduler which determines routing and timing of data transfer from each device. Controller complexity greatly increases as more I/O ports and buffers are added to the system. Frequency of device monitoring is a major factor in determining controller size and speed. The number of devices and I/O buffering also dictate speed and size.

The communication link connects all computers and controllers in the system. The data speed of a communication link is expressed in bits per second (baud). Current commercial applications can reach 9800 baud. Using fiber optics, a 20,000-baud rate is possible. A wider bandwidth and higher quality connecting cable must be used as speed increases. In addition, buffering and message interpreters become more costly.

Two basic parameters can describe HCS resources in relation to process control requirements: (1) maximum interrupt I/O processor speed and (2) resident memory available for controlling logic operations. The interrupt I/O speed is the maximum number of interrupts a processor can service at a specified I/O rate. This processor speed enables the processor to control the computer operating

system (O/S) and logic program execution without losing real-time data information exchange. The interrupt I/O speed can be modified and increased by adding buffers and multiplexers. Production process control requirements can be expressed in terms of I/O interrupt speed. The maximum number of interrupts per second of the process device is multiplied by the I/O rate requirements. This value gives a relative load on the computer's processor controlling the device.

The manufacturing process memory requirements depend on the controller used for the process device. For instance, a DNC machine with BTR would require more memory for control than a CNC machine. Since the entire part program has to be sent to the DNC machine instruction by instruction, the part program must reside in memory for fast I/O response. The CNC machine requires only a single part type instruction.

The interrupt I/O rate and available control memory define the type of computer needed to control the manufacturing process. Thus, the cost of meeting the manufacturing requirements can be obtained and is directly related to the production process. These two variables provide a simple representation of the control system. These parameters form the basis of a mathematical model to benchmark an HCS.

Hierarchical Control System Model

An HCS can be modeled as a resource allocation problem. Speed and storage capacity are the two resources to be allocated at each tier in the control system. The computer type selected for each tier will constrain the speed and memory capacity. Resource requirements are

based on the I/O speed, processor speed, and memory capacity needed to perform the computer's control functions. This model minimizes total system cost by determining the cost of speed and storage requirements at each level.

Tactical Controllers

The development of a hierarchical model begins at the process level. Machine requirements evaluations are complex and must be based on comprehensive knowledge of all possible production operations. By starting with the process level, the model is representative of the actual manufacturing process as well as the HCS itself. All production processes must be defined and NC machine requirements determined. Operation programs must also be developed for each machine. Once all machines N are selected, a controller option i must be selected for each NC machine j.

There are three major NC controller categories I: (1) DNC with BTR, (2) DNC with full program buffer, and (3) CNC. Although there are many variations, most controllers can be put into one of these major categories. DNC with BTR requires the most computer supervision. Controllers in this category usually have a small buffer storing only one or two program instructions. As the DNC machine executes one instruction, another instruction must be sent down from the controlling computer. The DNC machine with BTR is the least expensive controller. However, this type of control requires that part programs be stored in the throughput computer. In addition, extra computer memory and processor time are required to service DNC interrupt requests.

DNC with program buffer is similar to DNC with BTR, with one major enhancement. Under this type of control, the entire process program can be downloaded from the control computer into buffer. This method reduces storage and processor overhead on the throughput computer. If the DNC machines operate with adaptive control, the throughput computer must provide logic control decisions and send response signals back to the DNC adaptive control unit. DNC with adaptive control thereby increases the load on the throughput computer.

CNC machine controllers are micro- or minicomputers containing all machine process programs. In addition, CNC control units can service adaptive control logic decisions. CNC machine controllers are the most expensive. However, CNC machines greatly reduce the throughput computer's overhead, since demand on the throughput computer is limited to monitoring part status information.

Throughput computer I/O and interrupt requirements for each machine j under controller option i are designated r_{ji} and defined in kilobytes per second. Similarly, random-access memory (RAM) requirements are designated s_{ji} and expressed in kilobytes.

The total cost for implementing any combination of tactical controller options is:

$$\sum_{j=1}^{N} \sum_{i=1}^{I} C^{(1)}_{ji} x_{ji}$$

where:

$C^{(1)}_{ji}$ - cost of controller option i for process device j

x_{ji} - $\begin{cases} 1, \text{ control option i is used on device j} \\ 0, \text{ otherwise} \end{cases}$

N - total number of j devices

I - total number of i options

Only one option per machine can be selected:

$$\sum_{i=1}^{I} x_{ji} = 1 \text{ for all } j=1,\ldots,N$$

These expressions can establish speed and memory requirements as well as costs for each possible machine controller j under option i. With basic controller requirements directly related to the manufacturing process, the HCS can be defined.

Throughput Computer

The throughput computer monitors and controls part transfer, storage, and machine operations. This computer can provide a backup for the NC machine controllers. The number of throughput computers K depends on the type of computer option m and the number of machine controllers. There can be as many throughput computers as there are machine controllers or none at all. For the model, the number of computers is selected and the hierarchical system evaluated.

Once the number of computers is determined, controller tasks must be distributed among the computers if there is more than one throughput computer. Usually a computer k can be assigned to a group of tactical controllers J_k on the basis of memory and I/O interrupt rate. This balances the task load distribution such that:

$$\sum_{k=0}^{K} J_k = N \qquad k\varepsilon \{0,1,\ldots,N\}$$

$$J_k \text{ such that } \sum_{j_1=1}^{J_1} \sum_{i=1}^{I} r_{ji} s_{ji} x_{ji} = \sum_{j_2=1}^{J_2} \sum_{i=1}^{I} = r_{ji} s_{ji} x_{ji} =$$

$$\ldots = \sum_{j_k=1}^{J_k} r_{ji} s_{ji} x_{ji}$$

This task balancing technique does not have to be used. For instance, the machines could be assigned to production cells handling a common task. Machines might also be assigned by software requirements. Part programs will be less complex if they are developed on a single operating system. Machine locations could also dictate computer k assignment to machines J_k.

After machine assignment J_k has been determined, maximum I/O interrupt and memory capacities are evaluated for each computer option m. Each computer option m relates to hardware under consideration. The options can range from a microprocessor to a 64-bit mainframe. The maximum I/O interrupt speed R_{km} which computer k can provide under option m allows computer k to function on logic routines and to service peak interrupt requests. Similarly, maximum memory S_{km} is available for control and I/O under option m. These maximum I/O interrupt speeds and memory capacities are the amount of computer resource k available under option m, yielding the following constraints:

$$\sum_{j=1}^{J_k} \sum_{i=1}^{I} r_{ji} x_{ji} \leq \sum_{m=1}^{M} R_{km} y_{km} \text{ for every } k$$

$$\sum_{j=1}^{J_k} \sum_{i=1}^{I} s_{ji} x_{ji} \leq \sum_{m=1}^{M} S_{km} y_{km} \text{ for every } k$$

$$\sum_{m=1}^{M} y_{km} = 1 \text{ for every } k$$

where:

$$y_{km} = \begin{cases} 1, \text{ control option m is used for computer k} \\ 0, \text{ otherwise} \end{cases}$$

Thus, controller requirements are constrained by available I/O interrupt service speed and memory.

The computer cost $C^{(2)}_{km}$ includes system installation, operating system control, software development, hardware, and peripherals. Costs are determined for each computer k under each option m.

Computer/controller interface and peripheral storage costs C_{kmji} under options i and m cover the communications, multiplexers, ports, and data storage systems required to interface controller i with computer k under options k and m. The cost for the throughput computers is:

$$\sum_{k=0}^{K} \sum_{m=1}^{M} y_{km} \left(C^{(2)}_{km} + \sum_{j=1}^{J_k} \sum_{i=1}^{I} C_{kmji} x_{ji} \right)$$

Many throughput computer configurations can be included in the model. For example, communications among throughput computers can be allocated by software and hardware interface costs. If a tactical controller is not desired, then administrative and overhead computer

resource requirements can be removed from computer k's available resources.

Strategic Computer

When employed, the strategic computer provides long-range production output information, stores all programs required by throughput computers, backs up throughput computers which fail, and performs CAD and process simulation. The strategic computer has both an I/O interrupt speed $R^{(1)}_{l}$ and a memory capacity $S^{(1)}_{l}$. These computer resources must be greater than or equal to the resource demands from the throughput and tactical computers.

For the model, lower level demand is the sum of throughput computer resources R_{km} and S_{km}. Although this is a conservative estimate, the strategic computer could back up throughput computers in the event of total throughput-level computer failure.

In addition to the lower level requirements, the strategic computers also have overhead processor speed ρ and memory σ requirements not related to the actual manufacturing process. Thus, I/O interrupt speed $R^{(1)}_{l}$ and memory capacity $S^{(1)}_{l}$ for the strategic computer under option l are expressed as:

$$\sum_{l=0}^{L} R^{(1)}_{l} z_{l} \geq \sum_{k=0}^{K} \sum_{m=1}^{M} R_{km} y_{km} + \rho$$

$$\sum_{l=0}^{L} S^{(1)}_{l} z_{l} \geq \sum_{k=0}^{K} \sum_{m=1}^{M} S_{km} y_{km} + \sigma$$

where:

$$z_1 = \begin{cases} 1, & \text{option 1 is used for strategic computer} \\ 0, & \text{otherwise} \end{cases}$$

Another constraint added to determine whether the strategic computer is required can be expressed as follows:

$$z_o \geq R_{max} / \left(\sum_{k=0}^{K} \sum_{m=1}^{M} R_{km} y_{km} \right)$$

where:

R_{max} - maximum I/O, interrupt, processor capacity for strategic computer level

If the k throughput computers have the processing speed for the strategic level, a strategic computer is not required ($z_o = 1$).

Two costs are associated with the strategic computer. The first cost C_1 is the fully supported computer system under option 1. This cost includes hardware and software implementation and operation. The second cost C_{1km} is for interfacing throughput computer k under option m with the strategic computer under option 1. Also included is the cost F to store all programs on a peripheral memory device. Strategic computer costs are:

$$F + \sum_{l=0}^{L} z_l \left(C_l + \sum_{k=0}^{K} \sum_{m=1}^{M} C_{lkm} y_{km} \right)$$

When z_1 is equal to zero, a strategic-level computer is not required.

Hierarchical Model

The HCS mathematical model is presented in Figure 3.4. The hierarchical control implementation costs are contained in the objective function Φ. The objective function contains three sets of

$$\Phi = F + \sum_{l=0}^{L} z_l \left(C_l + \sum_{k=0}^{K} \sum_{m=1}^{M} C_{lkm} y_{km}\right)$$

$$+ \sum_{k=0}^{K} \sum_{m=1}^{M} y_{km} \left(C^{(2)}_{km} + \sum_{j=1}^{J_k} \sum_{i=1}^{I} C_{kmji} x_{ji}\right) + \sum_{j=1}^{N} \sum_{i=1}^{I} C^{(1)}_{ji} x_{ji}$$

subject to:

$$\sum_{k=0}^{K} J_k = N \qquad k \in \{0,1,\ldots,N\}$$

J_k such that $\sum_{j_1=1}^{J_1} \sum_{i=1}^{I} r_{ji} s_{ji} x_{ji} = \sum_{j_2=1}^{J_2} \sum_{i=1}^{I} = r_{ji} s_{ji} x_{ji} =$

$$\ldots = \sum_{j_k=1}^{J_k} r_{ji} s_{ji} x_{ji}$$

$$\sum_{j=1}^{J_k} \sum_{i=1}^{I} r_{ji} x_{ji} \leq \sum_{m=1}^{M} R_{km} y_{km} \quad \text{for every } k$$

$$\sum_{j=1}^{J_k} \sum_{i=1}^{I} s_{ji} x_{ji} \leq \sum_{m=1}^{M} S_{km} y_{km} \quad \text{for every } k$$

$$\sum_{l=0}^{L} R^{(1)}_{l} z_l \geq \sum_{k=0}^{K} \sum_{m=1}^{M} R_{km} y_{km} + \rho$$

$$\sum_{l=0}^{L} S^{(1)}_{l} z_l \geq \sum_{k=0}^{K} \sum_{m=1}^{M} S_{km} y_{km} + \sigma$$

$$z_o \geq R_{max} / \left(\sum_{k=0}^{K} \sum_{m=1}^{M} R_{km} y_{km}\right)$$

$$\sum_{i=1}^{I} x_{ji} = 1 \quad \text{for all } j=1,\ldots,N$$

$$\sum_{m=1}^{M} y_{km} = 1 \quad \text{for every } k$$

where: $\underline{x},\underline{y},\underline{z}$ are 0,1 integers

Figure 3.4. Hierarchical Control Mathematical Model

Boolean variables (x_{ji}, y_{km}, z_l). These variables indicate which hardware option is being considered for a specific computer or controller. Coupling costs are also included for each hardware option selected. Total system design and implementation cost is a function of the different hardware options available at each level.

Objective function costs (Figure 3.5) are subject to hardware constraints and production process requirements. The model assumes that machine process requirements have been determined and the number of throughput computers has been specified. Processing load J_k is distributed evenly among the selected throughput computers. As mentioned before, tactical controllers can be assigned by other methods (e.g., production cell or hardware capability). Once the tactical controller processing/memory load is distributed among the throughput computers, the feasibility of selecting a certain computer option at a hierarchical control level is determined by processor speed and memory requirements at the particular level.

A set of control hardware has a system cost defined by the objective function Φ. Thus, the HCS can be described as a minimum-cost resource allocation problem. The production process controllers require computer processor and communication speed and memory. Hardware selection determines the computer/controller speed and capacity available. Costs are based on the hardware selected. The hierarchical control model captures the basic costs and technological requirements for a flexible manufacturing system.

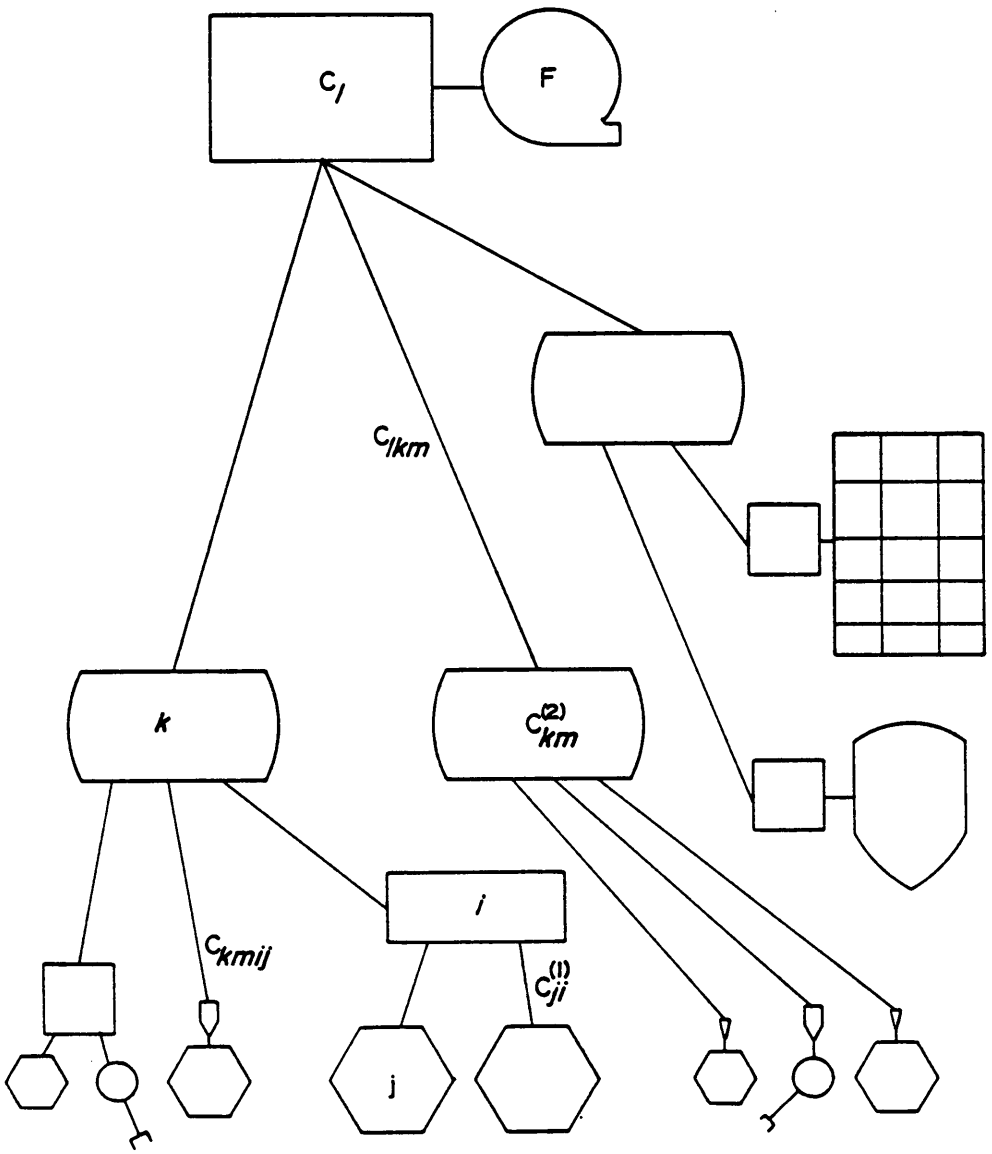

Figure 3.5. Hierarchical Control Costs

Model Analysis

A minimum-cost HCS can be identified with the mathematical model. The minimum-cost solution is a function of the production process and hardware control systems available. A general solution procedure is developed using dynamic programming techniques. A simple flexible manufacturing system is analyzed to illustrate the HCS model's utility.

Optimal Cost Solution

The possible number of feasible solutions to the model are finite. Only several hardware options are considered at each level. For an existing control system, the number of options is further reduced. Thus, there are discrete combinations of strategic, throughput, and tactical computers/microprocessors for the HCS.

The HCS model can be formulated as a dynamic program (Figure 3.6). States of the system are computing rate, capacity and memory requirements under the different computer and controller options. The stages are the HCS levels. Starting with stage 3, the entering state S_3 is the total system program storage requirement or task load. By selecting a number of throughput computers K, total system storage costs F are computed. The controller load distribution J_k on each throughput computer k is the state of system S_2 emerging from stage 3 and entering stage 2. At stage 2, the cost Φ_3 of throughput computer implementation is determined by the computer option y_{km} selected and the load distribution state S_2. For each option m selected, two states $\underline{S}_1\langle J_k\rangle$ and $\underline{S}_{11}\langle\cdot\rangle$ diverge from stage 2. State $\underline{S}_1\langle J_k\rangle$ is the maximum I/O interrupt processor rate and memory capacity to support

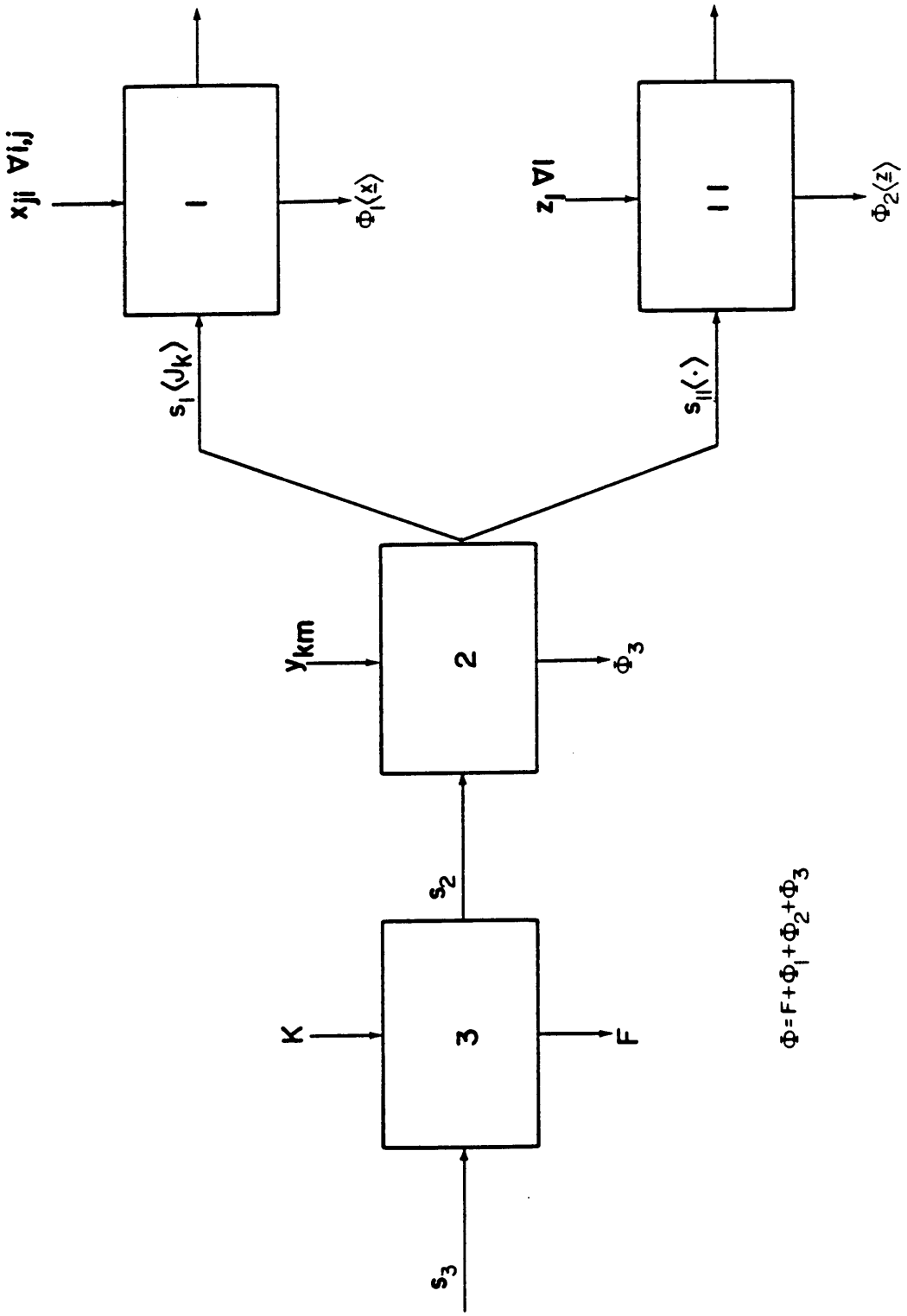

Figure 3.6. Hierarchical Control Dynamic Program

tactical controls assigned to computer J_k. The second diverging state $\underline{S}_{11}\langle\cdot\rangle$ is the maximum system and throughput computer processor, memory and storage requirement for strategic computer z_1.

At stage 1, feasible controller option costs $\Phi_1\langle\underline{x}\rangle$ are determined for the processor and memory capacities of the throughput computer state $\underline{S}_1\langle J_k\rangle$. Similarly, strategic computer costs $\Phi_2\langle\underline{z}\rangle$ are computed. For the processor speed and memory capacity requirements specified in state $\underline{S}_{11}\langle\cdot\rangle$, feasible z_1 options are determined along with the associated costs $\Phi_2\langle\underline{z}\rangle$. Summing all the returns from each stage provides the HCS total cost Φ.

In order to compute feasible solutions at stages 1 and 11, the number of throughput computers must be specified. In addition, throughput computer y_{km} must be selected. Once the throughput computers are defined, stage 1 is decomposed into k subproblems having the following form:

$$\begin{aligned} \text{minimize:} \quad & F_1 = C_x^T \underline{x}_k \\ \text{subject to:} \quad & \underline{r}_1^T \underline{x}_k \leq R_k \\ & \underline{s}_1^T \underline{x}_k \leq S_k \\ & \underline{r}_1^T, \underline{s}_1^T \geq 0 \end{aligned}$$

A minimum feasible cost is computed for each throughput computer k, using the throughput computer capacity limitation and controller requirements. Thus, given the entering state $\underline{S}_1\langle J_k\rangle$ under option m, the optimal feasible controller solution is determined.

At stage 11, the processor speed and memory capacity requirements defined for $\underline{S}_{11}\langle\cdot\rangle$ are defined for the strategic computer. The optimal solution can be found by optimizing the following subproblem:

$$\text{minimize:} \quad F_2 = C_z^T \underline{z}_k$$

$$\text{subject to:} \quad \underline{r}_2^T \underline{z}_k \geq R'_k$$

$$\underline{s}_2^T \underline{z}_k \geq S'_k$$

$$z_o \geq R_{max}/(\sum_{k=0}^{K} R'_k)$$

$$\underline{r}_2^T, \underline{s}_2^T \geq 0$$

Each option 1 is evaluated with respect to the throughput computer control requirements $\underline{S}_{11}\langle\cdot\rangle$. A minimum cost is computed based on the system requirements R_{max} and option 1 computer capacity.

Thus, minimum HCS costs are determined by a recursive technique, starting from stages 3 and 2. Given the number of throughput computers k, a feasible minimum objective function Φ value can be found and the resulting hierarchical system configuration determined. Again, the number of throughput computers can be selected, starting from a single throughput computer to a throughput computer for each processing device.

Example Problem

The following example problem illustrates the model's ability to benchmark HCS controller and computer requirements. The manufacturing system being analyzed is relatively small to illustrate the model's solution procedure. The costs are realistic and were obtained from actual hierarchically controlled flexible manufacturing systems.

A small manufacturing facility produces specialized parts. The parts are made in small batches. The types of NC machines required to meet production schedules have been determined. The system includes eight NC machines, two robots, an automatic storage and retrieval system (ASRS), and an automatic guided vehicle system (AGVS). The basic process design is presented in Figure 3.7.

Four computer types are considered for the HCS: an 8-bit microcomputer, 16-bit (small) minicomputer, 32-bit (large) minicomputer, and 64-bit mainframe computer. Data are collected on estimated computer and controller costs and reduced into model format. All labor, hardware, and software costs are included in the model analysis.

The model is used to evaluate HCS cost with zero, two, and three throughput computers (Tables 3.1 through 3.3). Optimal feasible solutions are found by evaluating each k and z_1 limit. When all the limiting constraints are satisfied, the feasible solution is identified by selecting the minimum-cost machine controller option. A three-tier computer system with a CNC control option is the optimal cost solution for this example (Table 3.3). This system comprises four small minicomputers at a cost of 5.410 million dollars. The system without a throughput computer (Table 3.1) requires a large minicomputer, while the system with two throughput computers (Table 3.2) requires three large minicomputers. The addition of a third throughput minicomputer enables a small minicomputer to replace the large strategic minicomputer, greatly reducing the cost. Since programming is a major

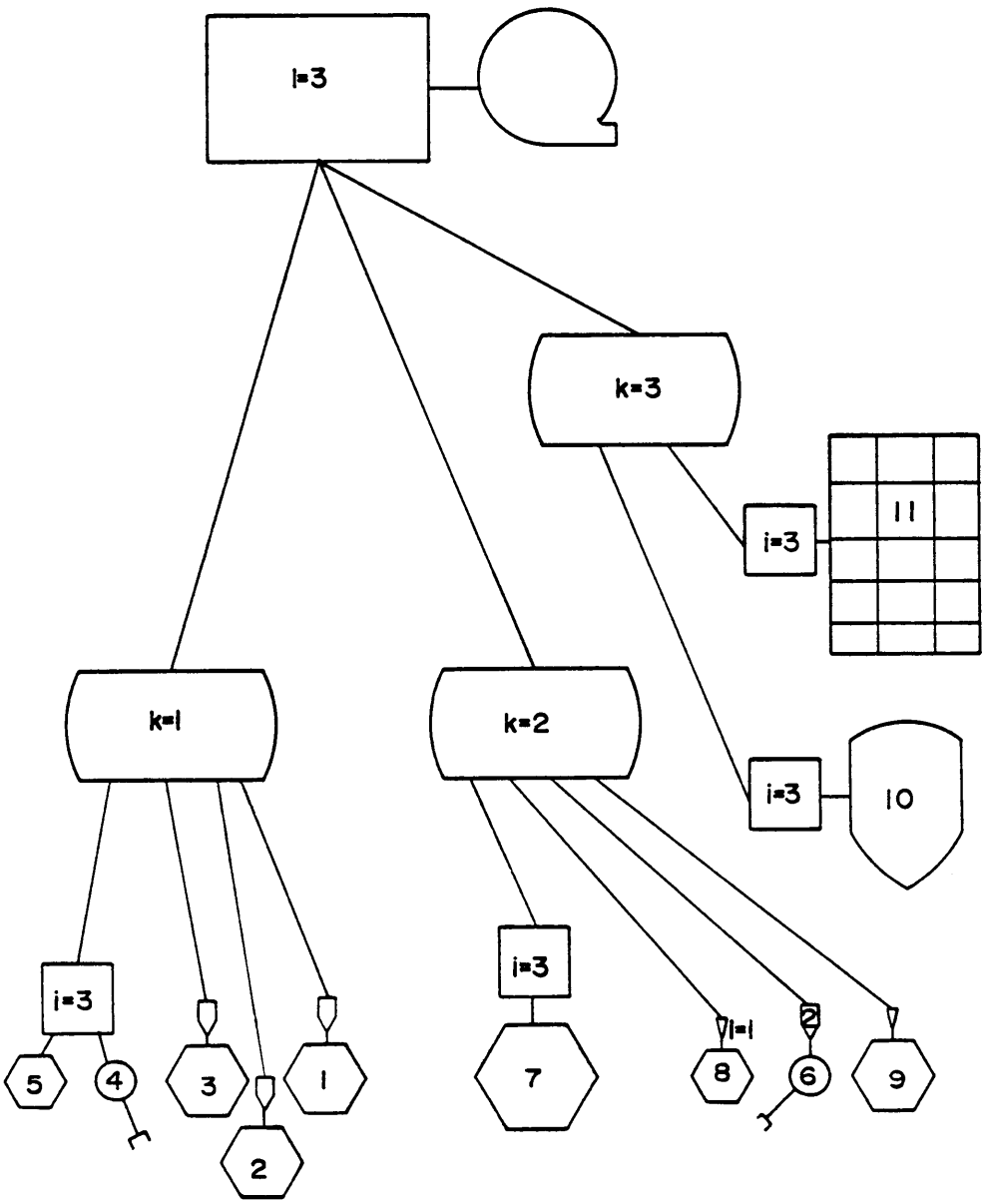

Figure 3.7. Hierarchical Control System Example

cost component of HCS implementation, controller options remain the same, due to the substantially higher programming requirement for DNC options. In addition, control system reliability is improved with a three-tier system.

Postoptimality analysis can be easily performed using the model output. For instance, if a CAD system were added to this example, a large time-sharing minicomputer might be required as the strategic computer. A large minicomputer without throughput computers is identified as the most cost-effective option for the modified configuration (Table 3.1), provided the memory requirements of the CAD system do not exceed 597 kilobytes. In a similar manner, other optimal control options can be analyzed without reevaluating the entire mathematical model.

Summary

The HCS mathematical model relates speed and capacity with system component cost. By minimizing the model's objective function, the most cost-effective HCS can be identified. The objective function is subject to technological and hardware limitations. The model's basic parameters can be modified to evaluate new technology. The HCS model can analyze all existing and future computer-controlled system designs to determine the most cost-effective computer control system for the specified manufacturing process.

The model results depend on the quality of the production data. Estimating costs and system operating requirements is a very difficult task. This model provides a general HCS configuration, using the

Table 3.1

Hierarchical Control System Without Throughput Computer

l	k	i	COST ($)$10^6$	SPEED (kbytes/s)	LIMIT (kbytes/s)	STORAGE (kbytes)	LIMIT (kbytes)
1	0	1	5.453	139.10	5	2088.00	0
		2	5.185	104.35		1452.00	
		3	5.018	10.65		778.50	
2	0	1	5.681	139.10	50	2088.00	176
		2	5.334	104.35		1452.00	
		3	5.160	10.65		778.50	
3	0	1	6.200	139.10	320	2088.00	1376
		2	5.837	104.35		1452.00	
		3	*5.693*	10.65		778.50	
4	0	1	9.845	139.10	720	2088.00	8500
		2	9.824	104.35		1452.00	
		3	9.670	10.65		778.50	

Table 3.2

Hierarchical Control System With Two Throughput Computers

l	m	i	COST $_6$ ($)10^6	SPEED (kbytes/s) k=1	k=2	LIMIT	STORAGE (kbytes) k=1	k=2	LIMIT	z_l SPEED	LIMIT	STORAGE	LIMIT
1	1	1	5.540	51.1	88	10	616	1472	32	20	5	64	0
		2	5.360	37.4	66.95		172	1280					
		3	5.071	10.65	16.7		10	768					
	2	1	5.643	51.1	88	68	616	1472	432	130		864	
		2	5.236	37.4	66.95		172	1280					
		3	5.176	10.65	16.7		10	768					
	3	1	6.203	51.1	88	260	616	1472	1500	520		3000	
		2	5.900	37.4	66.95		172	1280					
		3	5.726	10.65	16.7		10	768					
	4	1	9.439	51.1	88	740	616	1472	8400	1780		16,800	
		2	9.115	37.4	66.95		172	1280					
		3	9.018	10.65	16.7		10	768					
2	1	1	5.748	51.1	88	10	616	1472	32	20	50	64	176
		2	5.428	37.4	66.95		172	1280					
		3	5.299	10.65	16.7		10	768					
	2	1	5.871	51.1	88	68	616	1472	432	130		864	
		2	5.481	37.4	66.95		172	1280					
		3	5.360	10.65	16.7		10	768					
	3	1	6.400	51.1	88	260	616	1472	1500	520		3000	
		2	6.001	37.4	66.95		172	1280					
		3	5.897	10.65	16.7		10	768					
	4	1	9.600	51.1	88	740	616	1472	8400	1780		16,800	
		2	9.334	37.4	66.95		172	1280					
		3	9.122	10.65	16.7		10	768					
3	1	1	6.280	51.1	88	10	616	1472	32	20	320	64	1376
		2	5.912	37.4	66.95		172	1280					
		3	5.800	10.65	16.7		10	768					
	2	1	6.367	51.1	88	68	616	1472	432	130		864	
		2	5.987	37.4	66.95		172	1280					
		3	5.901	10.65	16.7		10	768					
	3	1	6.950	51.1	88	260	616	1472	1500	520		3000	
		2	6.476	37.4	66.95		172	1280					
		3	6.395	10.65	16.7		10	768					
	4	1	10.100	51.1	88	740	616	1472	8400	1780		16,800	
		2	9.762	37.4	66.95		172	1280					
		3	9.156	10.65	16.7		10	768					
4	1	1	9.760	51.1	88	10	616	1472	32	20	720	64	8500
		2	9.745	37.4	66.95		172	1280					
		3	9.722	10.65	16.7		10	768					
	2	1	9.876	51.1	88	68	616	1472	432	130		864	
		2	9.778	37.4	66.95		172	1280					
		3	9.615	10.65	16.7		10	768					
	3	1	10.435	51.1	88	260	616	1472	1500	520		3000	
		2	10.300	37.4	66.95		172	1280					
		3	10.100	10.65	16.7		10	768					
	4	1	13.300	51.1	88	740	616	1472	8400	1780		16,800	
		2	13.240	37.4	66.95		172	1280					
		3	13.201	10.65	16.7		10	768					

Table 3.3

Hierarchical Control System With Three Throughput Computers

l	m	i	COST ($)$10^6$	SPEED (kbytes/second) k=1	k=2	k=3	LIMIT	STORAGE (kbytes) k=1	k=2	k=3	LIMIT	z, SPEED	LIMIT	STORAGE	LIMIT
1	1	1	5.625	51.1	42	46	10	616	712	760	32	30	5	96	0
		2	5.335	37.4	32.6	34.35		168	640	680					
		3	5.175	10.4	8.15	8.6		10.5	382	386					
	2	1	5.767	51.1	42	46	72	616	712	760	432	195		1296	
		2	5.495	37.4	32.6	34.35		168	640	680					
		3	5.305	10.4	8.15	8.6		10.5	382	386					
	3	1	6.953	51.1	42	46	260	616	712	760	1500	480		4500	
		2	6.685	37.4	32.6	34.35		168	640	680					
		3	6.499	10.4	8.15	8.6		10.5	382	386					
	4	1	9.453	51.1	42	46	750	616	712	760	8500	2225		16,800	
		2	9.185	37.4	32.6	34.35		168	640	680					
		3	8.950	10.4	8.15	8.6		10.5	382	386					
2	1	1	5.801	51.1	42	46	10	616	712	760	32	30	64	96	176
		2	5.500	37.4	32.6	34.35		168	640	680					
		3	5.310	10.4	8.15	8.6		10.5	382	386					
	2	1	5.905	51.1	42	46	72	616	712	760	432	195		1296	
		2	5.734	37.4	32.6	34.35		168	640	680					
		3	5.410	10.4	8.15	8.6		10.5	382	386					
	3	1	7.081	51.1	42	46	260	616	712	760	1500	480		4500	
		2	6.678	37.4	32.6	34.35		168	640	680					
		3	6.660	10.4	8.15	8.6		10.5	382	386					
	4	1	9.118	51.1	42	46	750	616	712	760	8500	2225		16,800	
		2	8.834	37.4	32.6	34.35		168	640	680					
		3	8.660	10.4	8.15	8.6		10.5	382	386					
3	1	1	6.350	51.1	42	46	10	616	712	760	32	30	320	96	1376
		2	5.981	37.4	32.6	34.35		168	640	680					
		3	5.903	10.4	8.15	8.6		10.5	382	386					
	2	1	6.495	51.1	42	46	72	616	712	760	432	195		1296	
		2	6.127	37.4	32.6	34.35		168	640	680					
		3	6.061	10.4	8.15	8.6		10.5	382	386					
	3	1	7.675	51.1	42	46	260	616	712	760	1500	480		4500	
		2	7.295	37.4	32.6	34.35		168	640	680					
		3	7.156	10.4	8.15	8.6		10.5	382	386					
	4	1	9.725	51.1	42	46	750	616	712	760	8500	2225		16,800	
		2	9.337	37.4	32.6	34.35		168	640	680					
		3	9.263	10.4	8.15	8.6		10.5	382	386					
4	1	1	9.985	51.1	42	46	10	616	712	760	32	30	720	96	8500
		2	9.903	37.4	32.6	34.35		168	640	680					
		3	9.820	10.4	8.15	8.6		10.5	382	386					
	2	1	10.145	51.1	42	46	72	616	712	760	432	195		1296	
		2	10.125	37.4	32.6	34.35		168	640	680					
		3	9.970	10.4	8.15	8.6		10.5	382	386					
	3	1	11.346	51.1	42	46	260	616	712	760	1500	480		4500	
		2	11.320	37.4	32.6	34.35		168	640	680					
		3	11.170	10.4	8.15	8.6		10.5	382	386					
	4	1	13.145	51.1	42	46	750	616	712	760	8500	2225		16,800	
		2	13.020	37.4	32.6	34.35		168	640	680					
		3	12.870	10.4	8.15	8.6		10.5	382	386					

estimated cost. Before actual system design is implemented, a simulation analysis of the proposed system should be performed. The simulation results should provide a clearer insight into the control system interaction with the production process. In addition, hardware specifications can be made more exact. The mathematical model provides an excellent starting point for the simulation analysis. The mathematical model is a preliminary procedure for evaluating automated manufacturing systems.

The simulation model for evaluating the system is developed in Chapter 5. A simulation analysis will be performed for a basic cell identified by the mathematical model.

CHAPTER FOUR

ADAPTIVE CONTROL MATHEMATICAL MODEL

The basic structure of the hierarchical control model developed in Chapter 3 can be applied to any level of the manufacturing process. At the other end of the manufacturing computer control spectrum is adaptive control. An adaptive control system regulates the actual machine cutting process. The basic state-stage relationship developed for hierarchical control systems can be used to identify cost-effective component configurations for adaptive control systems. The optimal configuration must satisfy all operating requirements of an adaptive control system.

Adaptive control of machining processes can significantly increase productivity, improve workpiece quality, and reduce costs for many discrete part operations. Several studies have already shown that adaptive control can achieve significant reductions in metal machining time [18]. Adaptive control systems dynamically control machine parameters in response to variations in the machining environment, such as hard spots, dull tools or catastrophic failures. Machine process states are measured by sensors located at strategic points on the machine, in the cutting tool or on the workpiece itself. This type of control increases tool life and offers greater part protection during machining. The adaptive control systems being discussed optimize the machining process based on a predetermined performance index (e.g., tool life or production costs).

Adaptive control can greatly enhance the CIM process. In CIM, part programs are developed with a CAD system and sent directly to the CNC machine. CAD requires massive computer storage containing every possible cutting characteristic for each machining operation, tool characteristic, and workpiece material. Adaptive control reduces the requirements for machining data storage and decisions in a CAD system, increasing the efficiency of the design process. Machining cost reductions derived from adaptive control implementation can be completely offset by the cost of sensors and logic control systems. Thus, costs must be carefully considered in adaptive control system design.

In an adaptive control system, machine control decisions are based on the process states. These states are determined from sensor measurements of machining process parameters. For example, cutting force can be measured directly from strain gages or indirectly from the tool power transducer. Many types of sensors, microprocessors or minicomputers, and signal processing equipment must be considered in the design of an adaptive control system. Technological and cost tradeoffs must be made in adaptive control system design. For example, a decision must be made whether to use an indirect measurement technique with a high-speed microprocessor versus a more costly direct measurement sensor configuration and a less expensive processor. Thus, the problem is to determine the most cost-effective sensor configuration to measure the specified state parameters, given the maximum response time and accuracy needed to control the process.

A mathematical model is developed to determine a cost-effective adaptive control system for a given machining process. Hardware components are specified in terms of performance characteristics, such as sensor resolution, sensor signal analog-to-digital (A/D) conversion time, microprocessor cycle time, and machine servomotor response time. These hardware performance characteristics are expressed in general terms. Thus, many different technologies and configurations can be investigated together in the same model. New technologies, such as fiber optics, can be analyzed without changing the model's structure. Inputs to the model include a set of sensor configurations required to measure state variables for the proposed adaptive control system, costs and performance characteristics of the sensor and hardware components under consideration. Both constraint and optimal adaptive control systems can be evaluated with the model.

The model is constructed and solved as a resource allocation problem. Sensors and other hardware are the resources being allocated to measurement resolution and processing time requirements, based on the sensor configuration and machine servomotor response. Expressing the model as a serial multistage system, all major hardware components can be analyzed using dynamic programming to determine a minimum-cost configuration for the adaptive control system. The solution process uses hardware operating characteristics to reduce the enumeration required at each decision stage in the dynamic program. Postoptimal design analysis is easily performed due to the serial multistage structure. The solution procedure can also be evaluated with a

computer and used in real-time design analysis of adaptive control systems. The solution provides a minimum cost for each sensor configuration evaluated and the hardware components for the optimal adaptive control system.

Adaptive Control System

The basic adaptive control system for a metal removal process is presented in Figure 4.1. This design can be extended to other machining and manufacturing operations. Sensors measure machining parameters. A logic unit computes the machine process state from sensor data and determines control actions according to a performance index or predetermined state response. This system is different from a closed-loop system which compares measured data with the current control input signals. The control signal output from this comparison device is a predetermined function of the two signal inputs. The adaptive control system being evaluated uses digital logic to determine the control action. This is an important point. An optimization process can be performed more readily with a logic system than with a feedback control system [82]. In addition, adaptive control systems with the appropriate sensors can detect impending catastrophic tool failures and avoid them in most cases without stopping the entire machining process [75]. Usually feedback loops can detect a failure only after it has already occurred and stop the process completely. The digital system is more flexible since the controlling logic is software based. The feedback control algorithm is usually determined

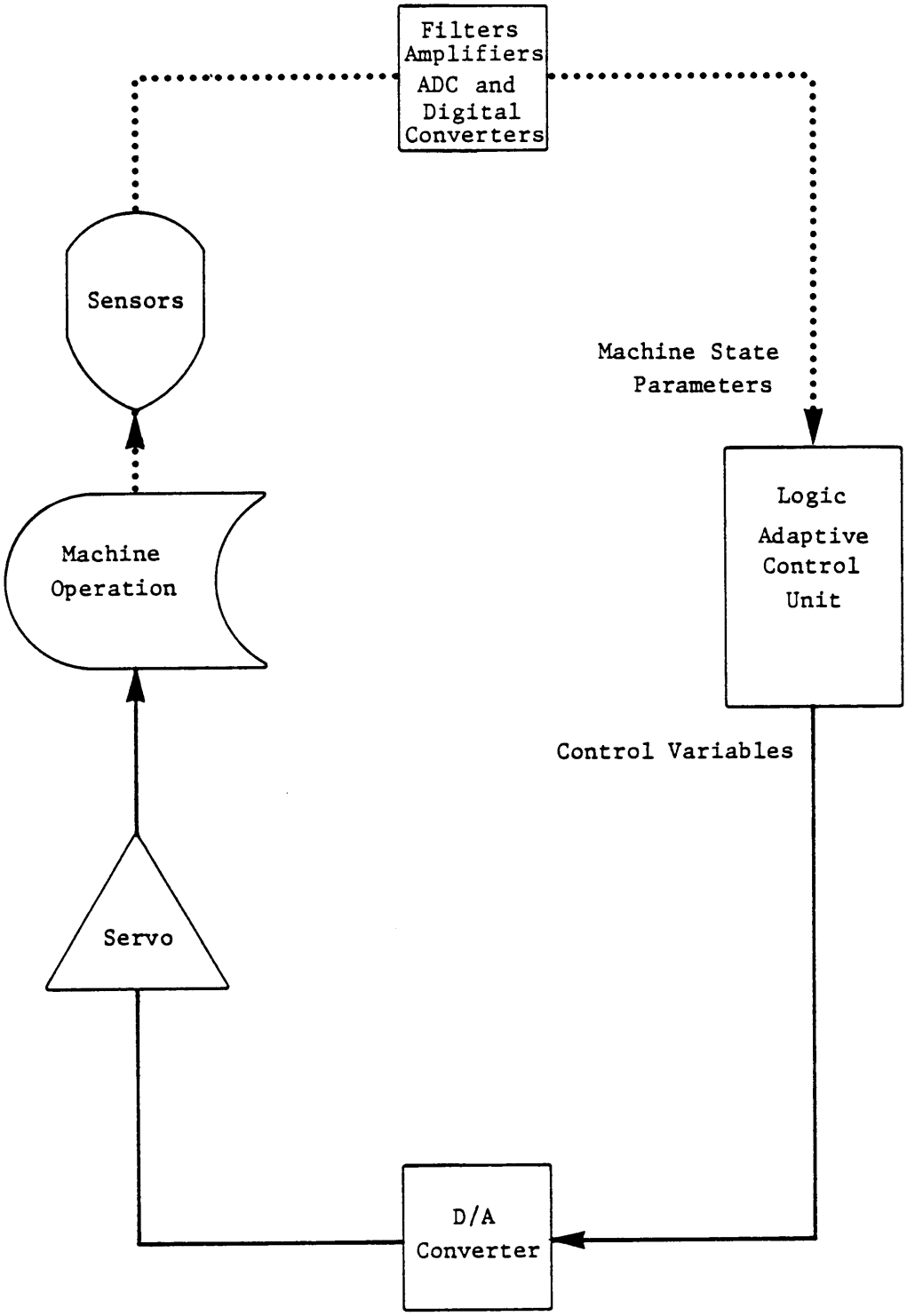

Figure 4.1. Adaptive Control System

by hardware components. Both the adaptive and feedback control systems are usually used in parallel. Each adaptive control system component and operating characteristic is discussed in the following paragraphs.

Sensors

Sensors in the adaptive control system measure the physical quantities of the machine process and convert them into either electrical voltage or current. The metal cutting environment is harsh for electrical transducers and other sensors. Most major studies on adaptive control of metal removal machining processes state that transducer reliability is the greatest problem in the adaptive control system [12, 53]. Noise, vibration, particulates, and other environmental disturbances from the machining process make measurements of the machining process extremely difficult. Overcoming these problems increases the cost of the sensors and must be considered in the adaptive control design analysis.

Adaptive control sensors can be divided into two major groups: analog and digital. Analog sensors produce a continuous voltage or current signal over time. Digital sensors produce a digital output signal. This signal can be a parallel set of digits representing a measured quantity or a pulse train that is counted over a specified time period. Some of the more common transducers used in metal cutting adaptive control systems are listed in Table 4.1 along with the measured physical quantity. On most CNC machines, the positioning and speed sensors are already installed on the machine and used for feedback control [35].

Table 4.1

Common Transducers for Adaptive Control Machine Systems

TRANSDUCER	MEASURED QUANTITY	OUTPUT
Ammeter	electrical current	voltage
Inductance-coil generator	rotational speed	pulse train
Linear-variable differential transformer (LVDT)	angular or linear displacement	voltage
Ohmmeter	electrical resistance	current or voltage
Photometric transducer	light intensity	current or voltage
Piezoelectric accelerometer	vibration	emf
Positive displacement flowmeter	fluid flow	pulse train
Potentiometer	voltage	current or voltage
Pressure transducer	pressure	voltage
Contact or proximity switches	on/off condition	voltage or current
Strain gages	torque, force, pressure, and other stress-related variables	voltage
Thermocouple	temperature	emf
Turbine flowmeter	flow rate	pulse train

The type of sensor selected depends on the machine system states required by the adaptive control system to determine control actions. Sensor specification is a very involved process, especially for analog devices. Many different transducer characteristics must be considered, including accuracy, repeatability, resolution, threshold value, calibration, drift, hysteresis, response speed, maximum measured quantity, and reliability. Of these characteristics, resolution, range, and threshold value can prescribe both analog and digital transducers in the initial design of a system [80]. Sensor costs can be directly related to these values.

Complete sensor packages are considered in the model. In addition to the transducer costs, gain amplifiers, filters, calibration hardware, feedback linearity and other signal processing equipment are included in the package. Digital sensor costs do not include pulse counters or the processor buffer interface.

Sensor/Processor Interface

Before the processor can perform logic operations on the sensor data, sensor signals must be converted into binary output which can be interpreted by the processor. Sensors produce three major types of signals: continuous analog data, discrete binary data, and pulse or discrete data. As discussed previously, the analog data are uninterrupted input signals over the measurement period. Discrete binary data have two possible levels (states): on (true) or off (false). These data represent a switch or relay contacts in the machining process. The data are represented by either high or low

voltage with respect to time. These voltage levels are assumed to change "instantaneously" with respect to time. Pulse or discrete data are not restricted to two input states. Pulse signals are a set of electrical pulses that can vary with time. Discrete data are a parallel combination of binary bits representing a measurement value or system state. Table 4.2 presents some of the uses of these data types in the machining and manufacturing process.

The continuous analog signal must be converted into binary form which is read by the processor. For each analog signal input to the processor, a signal amplifier and an analog-to-digital converter (ADC) are required. The ADC converts the incoming signal amplitudes into binary values. An ADC samples the analog signal at a specific rate and requires a certain amount of time to convert the sampled signal into a binary value. This conversion time is directly proportional to the number of binary bits that define the signal amplitude. Greater quantization of the signal amplitude requires longer conversion times. This number of bits also defines the ADC resolution of the measured value. If more than one analog input signal is processed, a multiplexer can be used to sample signals from the various sensors with one ADC. If high-speed sensor data rates are required, a separate ADC can be used for each sensor. Both A/D conversion processes for several sensors are shown in Figure 4.2.

Pulse data can be converted into binary values by counting incoming pulses over a specific time period [4]. A binary counter is used in conjunction with a clock. When the counting period is over,

Table 4.2

Sensor Inputs and Controller Outputs for NC Machine Operations

ANALOG SIGNALS	INPUT	OUTPUT
Spindle angular velocity	X	X
Feed rate	X	X
Cutting temperature	X	
Cutting force or torque	X	
Power consumption	X	
Position	X	

DISCRETE BINARY SIGNALS	INPUT	OUTPUT
Tool engagement	X	
Cutting fluid	X	X
End stops	X	
Emergency shutdown		X

PULSE DISCRETE SIGNALS	INPUT	OUTPUT
Stepping motors	X	X
Spindle angular velocity	X	
Feed rate	X	X
Position	X	X

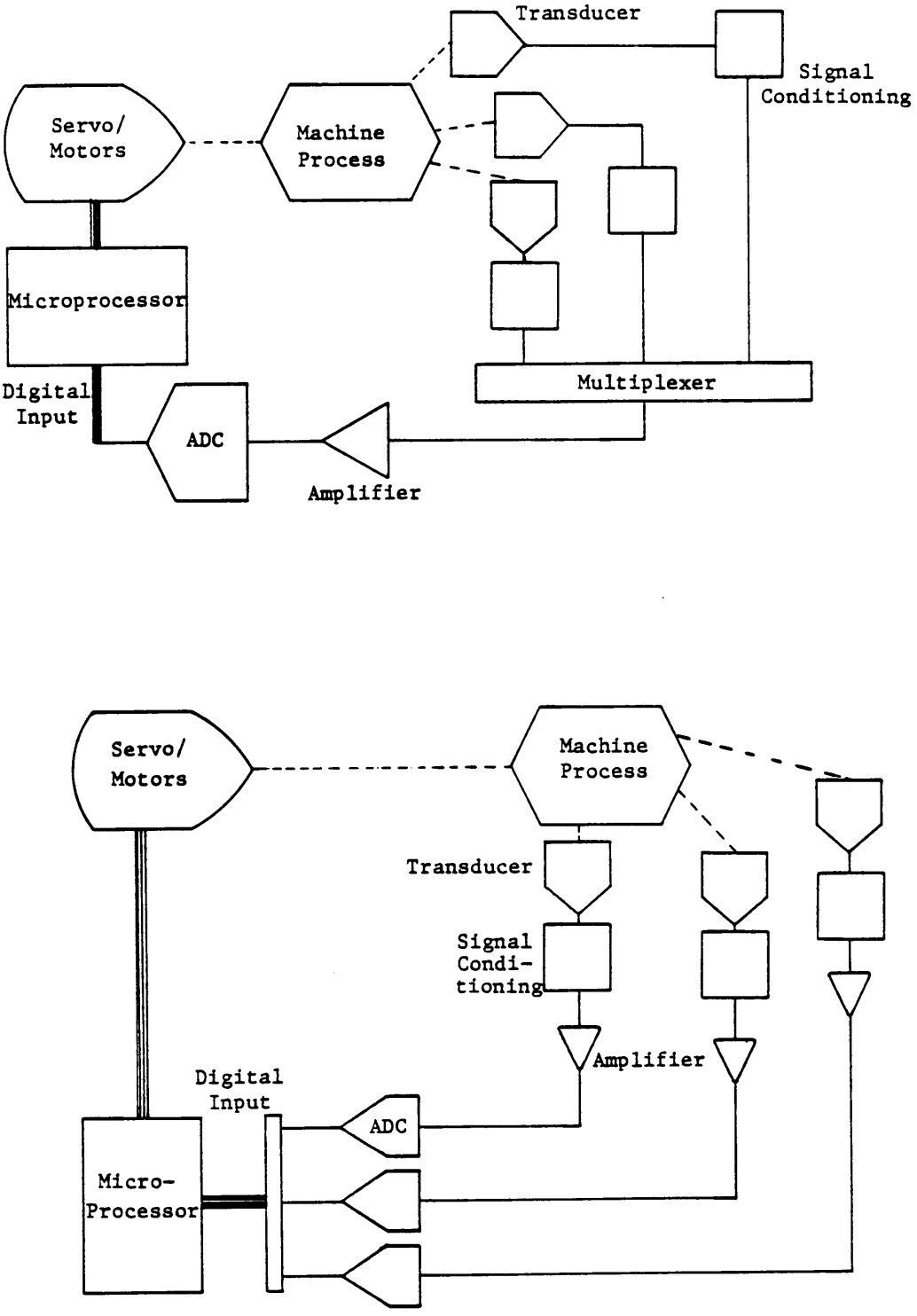

Figure 4.2. Analog-to-Digital Conversion

the binary count is read by the computer. Pulse data conversion hardware can also be expressed in terms of resolution and binary conversion period.

Discrete and binary data signals must be latched for the processor to read. Some binary inputs might also be connected to the processor interrupts. These signals can be read directly by the processor without time conversion hardware. Binary and pulse data are more compatible with the microprocessor and minicomputers and are easier to implement.

Except for the binary data, all sensor data conversion hardware can be specified in terms of resolution and conversion time. Sampling rates of most ADC hardware are much faster than most processes require. Sampling rates are less than 50 microseconds for a very slow ADC [3]. At this sampling rate, a high-speed drill spindle angular velocity can be measured more than six times per revolution at 1500 rpm.

Resolution is the change in physical quantity that each pulse represents (e.g., a fraction of a revolution or total linear displacement). A measurement resolution can be defined by:

$$\text{resolution} = \frac{\text{full-scale range}}{2^N}$$

where N is the number of bits in the ADC binary measurement register. The conversion period is the time required to obtain a count and prepare it for the processor to read. Signal conversion hardware is specified by conversion time and resolution in the model. These converter performance characteristics are related to hardware costs.

Processor

The adaptive control logic unit is a microprocessor or minicomputer which computes machine process states from sensor values. From the input states, the processor determines appropriate control decisions. These decisions require time to compute. This time depends on the sensor configuration, control strategy software, and processor. The processor execution time to input the data and compute control actions depends on operating speeds, register size, and arithmetic/logic unit (ALU) capabilities. Microprocessor clock cycles range from less than 1 MHz to 29 MHz [29]. Larger register sizes enable more data to be processed at a time. If the ALU can perform multiplication and division, control programs can be executed more quickly than in processors with only addition and subtraction capability. In addition, any special I/O support hardware can increase processing speeds.

The model considers the processor with memory, storage, and communication hardware as a single unit. General categories of controller boards are considered. This assumption simplifies processor selection and is consistent with the general practice of considering entire microprocessor systems when designing adaptive control systems. The processor's longest control program execution time for each sensor configuration must be estimated. This maximum processing time is used in the model analysis. Processor hardware and software requirements increase as a function of sensor information and control decision computations. In addition, faster memory access requirements could

dictate selection of a faster processor, increasing hardware costs. Software costs are a function of sensor configuration. These costs increase in direct proportion to the number of sensors and control program complexity associated with each configuration.

Machine Interface/Controller

Once the logic unit has made the control decision based on the state variables, control decisions must be sent to the appropriate servocontrollers, stepper motors, and/or relays. The processor sends a binary signal over a bus line to a specified device which converts the binary control word to an analog signal, pulse train or relay action. These devices convert the signal in the same manner as the sensor signal converters, except inputs and outputs are reversed (Table 4.2).

Binary control decisions from the microprocessor are changed into a continuous analog signal with a digital-to-analog converter (DAC). As with an ADC, a delay time exists. The resolution time depends on the input binary word. For the purpose of the model, it is assumed that the ADC and DAC have the same resolution. This assumption is made because the control actions have to be measured with the same resolution.

The pulse-train output is obtained by using a counter and a reference clock pulse. The number of pulses and rate over a given time period are sent from the processor as binary instructions. The counter sends the pulses out at the specific rate and the down counter is loaded with the number of pulses. When the down counter is finished counting, the pulse train is stopped. Binary data activates a

solid-state or electromechanical relay. The pulse-train data is specified by the conversion time into a pulse train from the digital instruction.

Finally, the motor or actuator response time machine drive system must be considered. This response time is the maximum amount of time required to change the machine from the current operating state to a steady-state operating condition specified by the controller. For example, a CNC machine might be able to change the spindle angular velocity by 5 rpm in 1500 milliseconds. Of course, changes in operating state are assumed to be linear over time. Most machines requiring adaptive control have a feedback control loop which maintains linear response control characteristics with respect to time.

This model assumes that all processor systems under consideration meet machine interface/control resolution requirements. A minimum machine adaptive control response time is specified for each sensor configuration. This response time is determined during the design phase of the sensor configuration and is a function of the machining process. Each machine control and motor system being evaluated is described by its maximum time to respond to an adaptive control signal. Thus, the minimum time remaining for the adaptive control measurement and decision cycle depends on the interface/control system and sensor configuration. The costs increase for interface/control systems with faster response times.

Adaptive Control System Design

A basic adaptive control configuration is illustrated in Figure 4.3. Sensors are selected and located on the machine to measure the machining process information required for determining system process states. Several different sensor configurations usually can provide information to determine the same adaptive control decisions. For example, tool wear might be detected by either a thermocouple placed in the tool or a strain gage on the tool holder [15, 76]. The thermocouple measures temperature increases as the tool flank wears. The strain gages can measure cutting force, which increases due to the tool wear. Each of these signals has different processing time and resolution requirements. The thermocouple signal might require higher signal resolution, greater linearization, and ADC conversion time than the strain gage signal. Computer processing time would also be different for the two input conditions. Temperature sensor response time might also be extremely slow for detecting changes in tool wear. Carbide tools have good heat capacitance, and diffusion to the tool thermocouple is delayed [15]. In addition, extra cost might be involved placing the thermocouple in each tool.

An adaptive control strategy must first be selected and input states identified. Once the control strategies have been defined, different sensor configurations must be specified for the system. The model assumes that the adaptive control strategy for each machine operation has been defined. Feasible adaptive control sensor configurations are identified. Performance requirements are determined

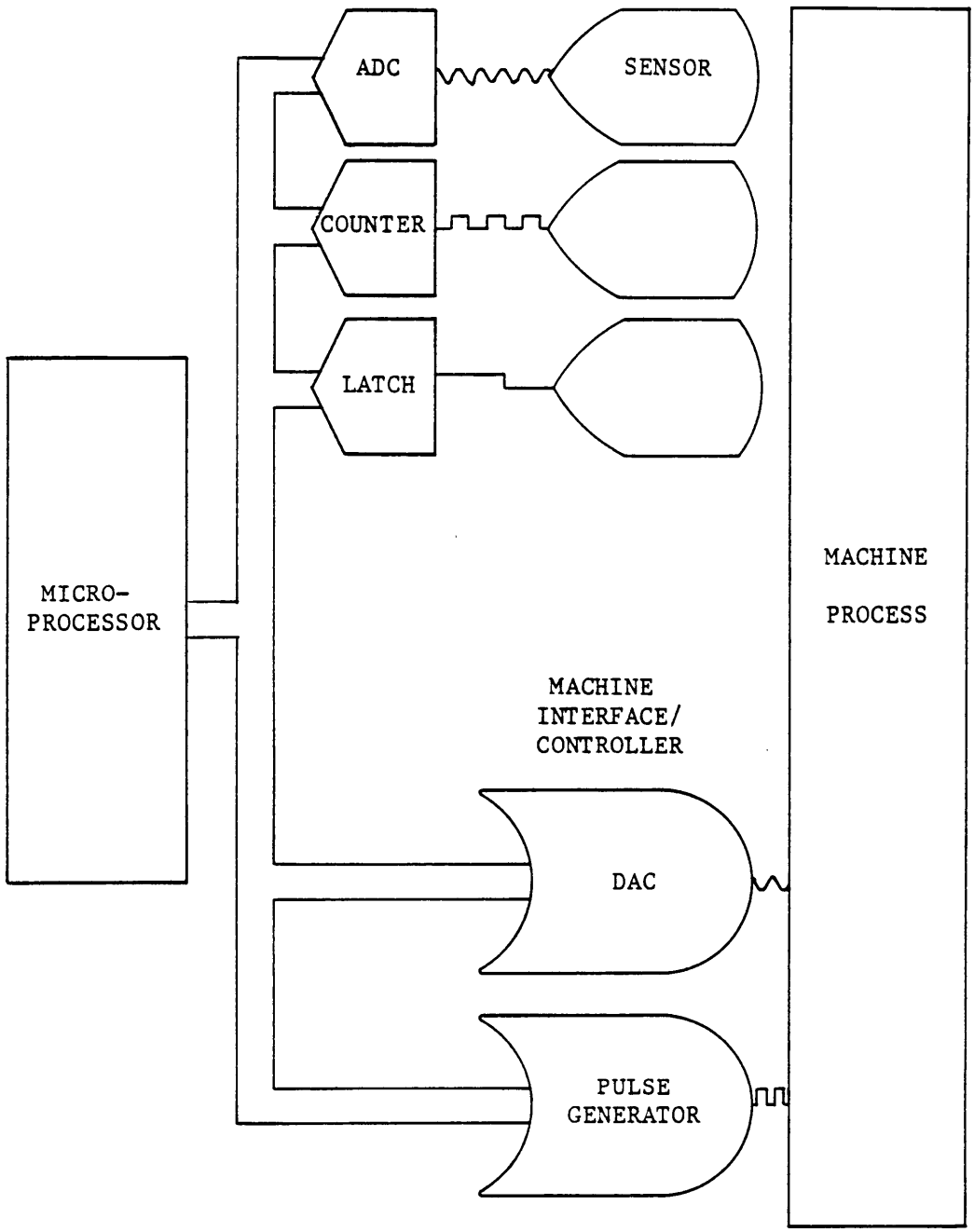

Figure 4.3. Hardware Configuration

for each type of hardware in the adaptive control system. Hardware components being evaluated must be expressed in terms of their performance characteristics. Component costs must be specified. These costs are a function of hardware performance. For example, a 16-bit microprocessor control board is more expensive than an 8-bit microprocessor controller. Once hardware costs and operating performance characteristics have been identified, the model analysis can be performed to determine the most cost-effective configuration, given the operating requirements of the adaptive control system.

Adaptive Control Evaluation Model

The model's technological operating constraints are developed in this section. The objective function relates costs to performance characteristic requirements.

Hardware Components

Four major components are analyzed by the model: (1) sensors, (2) sensor signal to processor hardware, (3) logic hardware, and (4) machine process control hardware. A selection variable is assigned to each component type. The variable takes on the value one if the component is selected. Otherwise the selection variable is assigned a zero. A specific sensor i must be selected for each adaptive control sensor type j specified in configuration k. Since only one configuration is selected, only one sensor can be selected for each sensor type required by that configuration. This constraint can be stated as follows:

$$\sum_{i=1}^{I_j} x_{kji} = 1 \quad \forall\, k, j \in J_k$$

where:

$x_{kji} = \begin{cases} 1, \text{ if sensor } i \text{ of type } j \text{ is selected for configuration } k \\ 0, \text{ otherwise} \end{cases}$

I_j - number of sensors of type j

J_k - sensor types in configuration k

One ADC or pulse-train converter l is selected for each sensor type j and is expressed as:

$$\sum_{l=1}^{L} z_{lj} = 1 \quad \forall\, j \in J_k$$

where:

$z_{lj} = \begin{cases} 1, \text{ if converter } l \text{ is selected for sensor group } j \\ 0, \text{ otherwise} \end{cases}$

L - number of converters l feasible for sensor group k

Only one configuration k can be selected for the adaptive control system. This constraint is stated as:

$$\sum_{k=1}^{K} v_k = 1$$

where:

$v_k = \begin{cases} 1, \text{ if configuration } k \text{ is selected} \\ 0, \text{ otherwise} \end{cases}$

K - number of adaptive control configurations under consideration

A single microprocessor or minicomputer n is selected for each configuration k. This restriction is expressed as:

$$\sum_{n=1}^{N} u_{nk} = 1 \quad \forall\, k$$

where:

$$u_{nk} = \begin{cases} 1, \text{ if processor n is selected with configuration k} \\ 0, \text{ otherwise} \end{cases}$$

N - number of processors under consideration

Finally, only one of the machine servo and motor control/ interfaces m under consideration can be chosen for the adaptive control configuration k. This constraint is stated as:

$$\sum_{m=1}^{M} w_{mk} = 1 \quad \forall\, k$$

where:

$$w_{mk} = \begin{cases} 1, \text{ if interface/control m is used with configuration k} \\ 0, \text{ otherwise} \end{cases}$$

M - number of interface/controls under consideration

(This constraint can be ignored if only one machine interface/control system is being considered.)

These constraints ensure that one adaptive control configuration with the required sensors and signal converters is selected by the model. In addition, the model selects only one processor and machine interface/control system for an optimal solution. As mentioned earlier, a set of sensors, adaptive control configurations, sensor converters, processors, and machine interface/control configurations must be determined in advance and used as input into the model.

Sensor Measurement Range

A measurement range is defined for each sensor type in the configuration monitoring the process. This range is determined by the adaptive control logic strategy. Each sensor i of type j has a

specific threshold quantity which can be measured. The sensor also has a maximum quantity which can be measured. Basic sensor performance characteristics are defined by these two values. The range defined here is for the entire sensor system with signal conditioning hardware and should not be confused with voltage or current ranges specified for transducers. For example, a thermocouple would be specified in terms of the temperature and not the voltage output. This is an important distinction since most transducer outputs are expressed in terms of voltage or current range.

Maximum (minimum) threshold and maximum (minimum) measurement quantities are specified for each sensor type in the configuration. This measurement range and threshold value specification is defined as a constraint and can be expressed as:

$$x_{kji} \cdot d_{min\ ji} \leq \sum_{k=1}^{K} D_{min\ kj} \cdot V_k \qquad \begin{cases} i \varepsilon I_j \\ j \varepsilon J_k \\ k=1,\ldots K \end{cases}$$

$$x_{kji} \cdot d_{max\ ji} \geq \sum_{k=1}^{K} D_{max\ kj} \cdot V_k$$

where:

$d_{min\ ji}$ - minimum measurement level for sensor i of type j

$d_{max\ ji}$ - maximum measurement level for sensor i of type j

$D_{min\ kj}$ - maximum threshold measurement value for sensor type j in configuration k

$D_{max\ kj}$ - minimum threshold measurement value for sensor type j in configuration k

This constraint defines the feasible measurement region for each sensor configuration being analyzed.

Resolution

The sensor signal must be converted into a binary number corresponding to the measured quantity. A specific resolution is required for each adaptive control sensor configuration. The sensor resolution must be less than the resolution required by the logic program. In addition, ADC and pulse-train converters must provide the processor with the appropriate resolution. The resolution is determined by the number of bits used to present the converted signal.

Thus, each feasible converter and sensor pair must have a resolution less than the specified resolution. The resolution is a function of the sensor range and number of converter quantization levels 2^N. This constraint can be expressed as:

$$\sum_{i=1}^{I_j} \frac{X_{kji} \cdot |d_{max\ ji} - d_{min\ ji}|}{2^{\left(\sum_{l=1}^{L} y_{lj} \cdot z_{lj}\right)}} \leq R_{kj} \quad \forall\ k,j$$

where:

y_{lj} - number of quantization levels l with sensor type j

R_{kj} - maximum resolution requirements for sensor j in configuration k

This constraint defines the feasible converter and sensor pairs. All possible pairs are considered. If a pair combination is infeasible, z_{lj} is zero, and an infinite value is computed for the resolution.

Time

An adaptive control system must, within a specified amount of time, bring a machine system from an existing state to a more desirable state defined by the control performance index and strategy. Thus, the hardware components, servos, and drive motors selected for the system must be able to meet the response time constraints as specified by the adaptive control sensor configuration. Minimum time requirements are determined by the type of machine interface/control system. Each processor under consideration is expressed in terms of maximum time to execute a control program and, finally, the time required to convert the sensor signal to a digital output.

The time constraint is stated as:

$$\sum_{k=1}^{K}\left[V_k\, w_{mk}\, \tau_{mk} + \sum_{j\epsilon k}\sum_{i=1}^{I_j} x_{kji}\left(\sum_{l=1}^{L}\alpha_{lj}\, z_{lj} + \sum_{n=1}^{N}\mu_{nk}\, u_{kn} + \beta_{ij}\right)\right] \leq T_m \quad \forall\, m$$

where:

α_{lj} - conversion time for sensor type j with converter resolution l

β_{ij} - sensor measurement cycle time

τ_{mk} - machine interface/control time under configuration k

μ_{nk} - maximum program time for processor n under configuration k

T_m - minimum adaptive control cycle time for control/interface m

Processor input time can be included in the converter time α_{lj} if this time is a significant part of the adaptive control cycle time. A significant amount of time might be required when the processor register size is less than the converter's binary measurement value. Again, the machine response time includes all interface/control delays.

The model assumes that the processor can control the machine. In the case of one machine interface/control component, only one response time has to be considered, thereby simplifying the problem. This is the case for most design problems. The machine servo and motors are designed first to specific machining requirements. The adaptive control system is then developed to optimize and/or maintain the machining process and prevent catastrophic failures.

Objective Function

Costs in the adaptive control system are directly related to the components, i.e., (1) sensor, (2) converter, (3) processor, and (4) machine controller. These costs reflect not only hardware, but installation and software development. The sensor cost depends on the configuration and the sensor selected. Converter costs are determined by the type of converter selected and its quantization level. Processor costs include all components on the logic controller board. This cost also includes software development and peripheral devices. Software costs depend on the configuration of the sensors. The more sensors interfaced with the controller, the higher the cost of the hardware and software control program. Thus, cost of the logic controller for the system is a function of the type of microprocessor and hardware used and the sensor configuration. Finally, the machine interface/controller cost is a function of the sensor configuration and the type of interface/control selected. If more responsive motors are selected, the cost of the motors increases. Each of these costs is discrete since individual pieces of hardware are being considered.

The objective function for the model can be expressed as:

$$\Phi = \sum_{k=1}^{K} \left[\sum_{m=1}^{M} C^3_{mk} w_{mk} + \sum_{n=1}^{N} u_{kn} C^2_{kn} + \sum_{j \epsilon k} \left(\sum_{l=1}^{L} C^1_{lj} z_{lj} + \sum_{i=1}^{J_k} x_{kji} C_{kji} \right) \right]$$

where:

C_{kji} - sensor cost

C^1_{lj} - signal conversion time for instrument j and converter l

C^2_{kn} - control processor cost

C^3_{km} - machine interface/control costs

System cost is determined by summing over all components under consideration and selecting a single optimal configuration. Each component cost is multiplied by an appropriate indicator variable. When the component is not considered, the indicator variable is zero. Otherwise, the hardware component is selected, and the cost is included in the objective function.

This final adaptive control model includes all the components in terms of their technological specifications. The system requirements are defined by the constraints. The objective function defines the cost of a system. The next step is to analyze the model and develop a solution method for determining a minimum-cost configuration given the component performance requirements.

The final adaptive control mathematical model is:

$$\Phi = \sum_{k=1}^{K} \left[\sum_{m=1}^{M} C^3_{mk}\, w_{mk} + \sum_{n=1}^{N} u_{kn}\, C^2_{kn} + \sum_{j \epsilon k} \left(\sum_{l=1}^{L} C^1_{lj}\, z_{lj} + \sum_{i=1}^{J_k} x_{kji}\, C_{kji} \right) \right]$$

subject to:

$$x_{kji} \cdot d_{\min\, ji} \leq \sum_{k=1}^{K} D_{\min\, kj} \cdot V_k \qquad \begin{cases} i \epsilon I_j \\ j \epsilon J_k \\ k = 1, \ldots K \end{cases}$$

$$x_{kji} \cdot d_{\max\, ji} \geq \sum_{k=1}^{K} D_{\max\, kj} \cdot V_k$$

$$\sum_{i=1}^{I_j} \frac{X_{kji} \cdot \left| d_{\max\, ji} - d_{\min\, ji} \right|}{\left(\sum_{l=1}^{L} y_{lj} \cdot z_{lj} \right)^2} \leq R_{kj} \qquad \forall\ k,j$$

$$\sum_{k=1}^{K} \left[V_k\, w_{mk}\, \tau_{mk} + \sum_{j \epsilon k} \sum_{i=1}^{I_j} x_{kji} \left(\sum_{l=1}^{L} \alpha_{lj}\, z_{lj} + \sum_{n=1}^{N} \mu_{nk}\, u_{kn} + \beta_{ij} \right) \right] \leq T_m \qquad \forall\ m$$

$$\sum_{m=1}^{M} w_{mk} = 1 \qquad \forall\ k$$

$$\sum_{n=1}^{N} u_{nk} = 1 \qquad \forall\ k$$

$$\sum_{k=1}^{K} v_k = 1$$

$$\sum_{l=1}^{L} z_{lj} = 1 \qquad \forall\ j \epsilon J_k$$

$$\sum_{i=1}^{I_j} x_{kji} = 1 \qquad \forall\ k, j \epsilon J_k$$

Model Analysis

The adaptive control model can be decomposed into a serial multistage system. At each stage, hardware components are selected for the adaptive control system. Hardware selection decisions are dependent on the state of the system and decisions made at preceding stages. With the model expressed in this serial structure, an optimal adaptive control system configuration can be determined using dynamic programming techniques. Special optimization procedures are employed at each stage. These procedures take advantage of the hardware performance characteristics and greatly reduce enumeration requirements. The entire model analysis can be implemented on a computer and used in a real-time design process. The decisions and transition functions are discussed for each stage of the serial system, along with optimization analysis.

Dynamic Programming Analysis

The serial multistage adaptive control system is presented in Figure 4.4. The system represents the entire design process, starting with the machining process at stage 4 and ending with the logic processor at stage 1. At stage 3, sensors are selected. At stage 2, converters are chosen. The input state S_4 is the sensor configuration k. The output state S_1 is the time remaining after a system has completed a maximum response cycle. Since this time must be greater than or equal to zero, the final state of the system is defined.

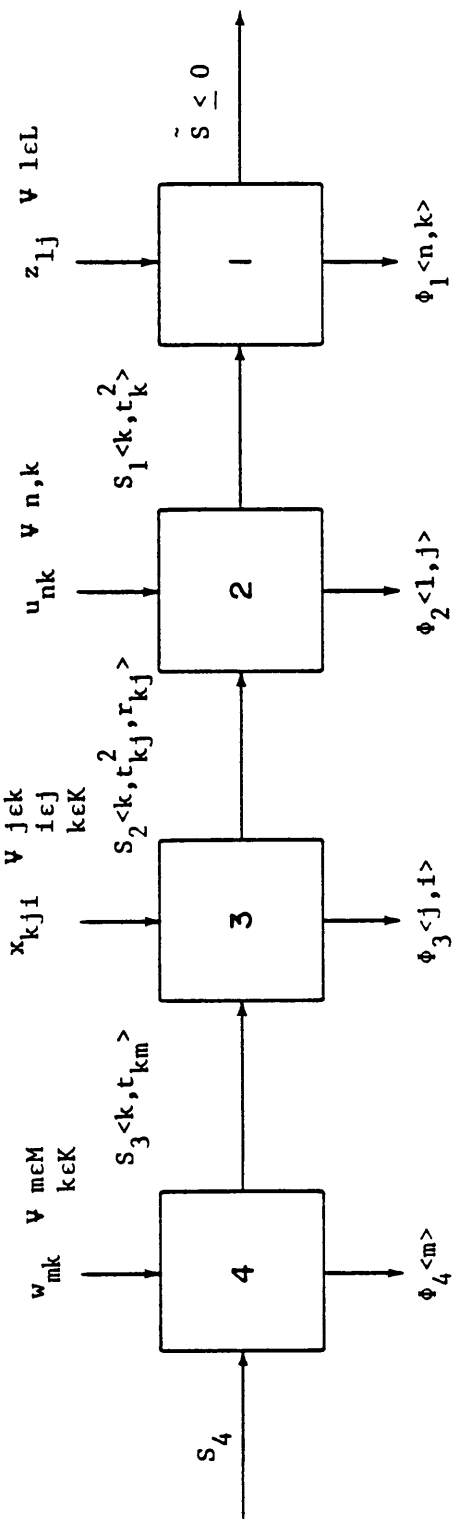

Figure 4.4. Serial Multistage System

Costs for each component in the adaptive control system are decoupled and are dependent on the stage decision and the input state. State transition functions at each stage are a function of the input state and/or stage decision. A minimum-cost solution can be found using dynamic programming. Since this is a final value problem, state inversion is used. The solution specifies minimum costs as well as the associated hardware components. Costs for the other sensor configurations are also provided because each input state S_4 is evaluated during the state inversion solution procedure. This solution aids in postoptimality analysis. A minimum-cost solution is found by beginning with stage 4 and optimizing each successive stage.

Stage 4: Machine Interface/Controller Selection

A machine interface/controller is determined at stage 4. Stage 4 enables the model to be directly integrated into the machine design process when different control motors are being selected. These interface/controllers determine the maximum allowable adaptive control system response times. The stage 4 input state is the sensor configuration. This state is passed directly to the next stage along with response time remaining for the adaptive control system t_{mk}. The response time variable depends on the machine interface/controller decision. The minimum-cost interface/controller system is selected for each sensor configuration k.

In most cases, the motor response times are determined for the machining process. Only one system response time is provided for each sensor configuration k. Thus, machine controller costs are the same

for each configuration in the analysis and can be ignored. Note that the response time given for each k is determined by the input state and is passed directly to the output state. If the same response time is used for every sensor configuration k, stage 4 can be eliminated and the remaining adaptive control response time is included directly in the stage 3 input state vector S_3.

Stage 3: Sensor Selection

Minimum-cost sensor configurations are determined at stage 3. For every configuration k, sensors i have to be selected for each sensor type j required by the configuration. These sensors must meet the threshold and range requirements of configuration k. Sensors under consideration for the adaptive control system can be arranged by sensor type j in order of decreasing cost. The threshold value can be checked first for feasibility with the sensor maximum threshold values for the configuration $R_{min\ kj}$. Next, those sensors meeting threshold requirements are compared with the minimum range requirement $R_{max\ ki}$. The minimum-cost feasible sensor is then selected for each sensor type j used in sensor configuration k. Costs of the sensors selected are added to the returns of the preceding stage for each configuration k.

Output states from stage 3 are the sensor configuration k, time remaining for converter and processor t_{mk}, and resolution requirements for each sensor type j in configuration k. Both the configuration and remaining cycle time are passed directly through the stage if the sensors do not add a significant amount of time to the cycle time.

If the sensors do add time, the maximum sensor recognition time of all the sensors in a configuration is subtracted from the remaining cycle time t_{mk}. Converter resolution requirements are determined by the sensors selected. This resolution is:

$$r_{kj} = \ln(d_{max\ ji} - d_{min\ ji})/(R_{kj} \cdot \ln 2.0)$$

Note that this resolution is dependent on the sensor decision. The resolution requirements for the optimal stage decision are in the output state vector to the next stage.

Stage 2: Converter Selection

At stage 2, both analog and digital converters are selected based on the input state. Although this appears to be an enormous task given the input state vector and the number of converters, the optimal decision for each configuration can be determined with very little difficulty. First, both conversion time and converter costs usually increase with the number of bits in the converter's output register. The converter with the lowest bit register should also have the minimum-cost and conversion time. Minimum-cost converters can be determined for each sensor type j in each sensor configuration k by selecting the lowest bit converter which meets input state r_{kj} resolution requirements. Once again, the converters can be arranged in increasing order of bit size. If one or more of the same type of converter (analog or digital) being considered have the same bit size, these converters are arranged according to least cost. Thus, a converter is selected for each sensor in a sensor configuration. Converter costs are added to the returns from the preceding stages.

The output state is a vector including the sensor configurations and the amount of time remaining in the cycle after sensor signal conversion t_k^2. The sensor configuration is again equal to the input configuration state ($S_2 = \tilde{S}_2$). Time remaining for each sensor configuration is determined by the converters selected. This time is the input state $\tilde{S}_2$ time remaining t_{mk} minus the maximum conversion time of all the converters in the configuration. Once the minimum-cost converters are selected, the maximum conversion time is the time remaining in each configuration for the processor and is included in the output state vector. Only the converters selected for each sensor have to be evaluated to determine the time remaining t_k^2.

Stage 1: Processor Selection

The processor for the adaptive control system is selected at stage 1. The input state to stage 1 is described by a two-dimensional vector $S_1(k,t)$. This vector contains the sensor configuration selected in previous stages and the time remaining t_k^2 to compute a control action after the sensor signal has been detected, converted, and sent to the processor. After receiving all the sensor signals, each processor has a maximum cycle time to compute a control action μ_{nk}. This maximum cycle time is a function of sensor configuration in the input state and the processor. The output state S_3 is the processing time remaining after the longest control cycle time has been completed by the adaptive control system. This time must be greater than or equal to zero. Otherwise, the adaptive control system cannot respond in the time T_m required for the adaptive control sensor configuration.

Thus, the processor's maximum cycle time must be less than the response time remaining t_k^2.

The processor cost is a function of the sensor configuration. Processor costs for each sensor configuration k are evaluated from lowest to highest cost. The first processor with a feasible processing time is selected. Once the minimum-cost processor is selected for each sensor configuration, the processor costs are added to the returns for the configuration. The sum of returns for each sensor configuration from the previous states is the minimum cost up to this final stage. The configuration having the lowest cost is a minimum-cost solution for the serial multistage system and the adaptive control system.

Postoptimization Analysis

The multistage system is structured for postoptimization analysis of the solution results. Sensor configuration is the input state to the serial system. Since this is a final value problem, minimum costs for each sensor configuration are determined directly from the dynamic program solution. Thus, each sensor configuration can be compared on a cost basis. For example, a minimum-cost configuration identified by the model might not be the most reliable configuration. The increased costs to implement a more reliable configuration can be obtained directly from the model. At each stage, reliability of individual hardware components can be evaluated. The effect of the component on total system cost is directly obtained from the model solution. For instance, a more reliable sensor might be substituted for a sensor identified in the minimum-cost solution. However, this sensor costs

more and has a lower delay measurement response. The sensor costs and performance data are identified in stage 3 and returns determined. At stage 2 the converter is determined. Finally, at stage 1 a processor is selected to account for the measurement delay. The returns from each stage are summed and added to the return from stage 4. The return is the minimum cost for the new component and can be compared with the previous minimum-cost solution. The cost of the increase in reliability can be analyzed. Reliability analysis is extremely important in adaptive control system design, since hardware failure is the greatest problem in adaptive control of machining processes.

Sensitivity analysis can also be used to design more cost-effective systems. Once the minimum-cost configuration has been identified, the sensor hardware performance requirements can be directly evaluated. Constraining performance requirements can be examined. The requirements can be lowered and cost changes evaluated. Thus, the decrease in system performance is compared with decreases in system costs. For example, a 16-bit processor is identified in the minimum-cost solution due to the amount of processing time required for the logic control program. The model solution indicates that if 55 microseconds are removed from the 8-bit logic program, total processor costs can be reduced by $800. The cost of reducing the program maximum execution time can be weighed against the decrease in adaptive control system cost. New component types can also be evaluated in the configuration. A different sensor type might require a shorter conversion time and reduce the processor costs. Model sensitivity

analysis allows many combinations of hardware and design changes to be evaluated without re-solving the complete problem.

Example Problem

The solution procedure and sensitivity (postoptimality) analysis are presented in a simple example problem. Two adaptive control sensor configurations optimally controlling tool life are compared and evaluated for a turning operation. Sensor configuration k=1 uses a power transducer and thermocouple to determine flank wear. Feed rate is also required for the control decision state and is obtained from a tachometer. The second configuration uses a strain gage, feed encoder, and spindle tachometer to determine flank wear and the process state. The six sensor types required for the sensor configurations are: thermocouple (j=1), strain gage (j=2), watt transducer (j=3), low-speed tachometer (j=4), high-speed tachometer (j=5), and LVDT (j=6). One machine interface/control system is being considered in the design analysis. Thus, the analysis begins with stage 3.

Analysis

The problem is structured into state and return solution tables (Figure 4.5). Beginning with stage 3, the input states are sensor configuration k and maximum time remaining for control cycle t_{mk}. These states are indicated in the figure. Sensors are arranged by increasing cost for each sensor type. The performance characteristics of each sensor are placed in the appropriate columns as shown in Figure 4.5. The threshold measurement value $d_{min\ ji}$ is compared with

k	j	i	$d_{min\ ji}$	$D_{min\ kj}$	$d_{max\ ji}$	$D_{max\ kj}$	C_{ijk}	Range	R_{kj}	r_{kj}	t_{mk}	t^1_{kj}
1	1	1 2 3	5 5 10	485°C	750 950 1250	945°C	345 362 389	945	1.0	10	250 ms	250
	3	1 2 3	0.1 0.5 0.5	2 hp	2 20 200	18 hp	215 225 237	19.5	0.1	8		250
	4	1 2 3	0.5 0.1 0.05	0.1 rpm	20 15 10	15 rpm	275 285 295 * r_3 = 872	15.9	0.1	8		250
2	2	1 2 3	50 185 370	50 lb	500 1000 5000	500 lb	1500 1525 1875	450	1	9	250 ms	250
	5	1 2 3	2 5 5	10 rpm	500 750 1000	800 rpm	150 160 165	995	1.0	10		250
	6	1 2 3	0.1 0.05 0.01	0.01 mm	20 10 5	5 mm	1850 2250 2750 r_3 = 4415	4.99	0.001	13		250

j = 1 thermocouple
j = 2 strain gage
j = 3 watt transducer
j = 4 low-speed tachometer
j = 5 high-speed tachometer
j = 6 LVDT

Figure 4.5. Model Analysis

k	j	l	r_{kj}	y_{1j}	C^1_{1j}	t^1_{kj}	α_{j1}	t^2_k
1	1	1 2 3	10	8 10 12	445 465 495	250 ms	10 μs 20 50	249.98
	3	1 2 3	8	8 10 12	295 300 395	250 ms	14 25 50	249.86
	4	1 2 3	8	8 10 12	385 375 395 $r_2^* = 1140$	250 ms	10 20 50	249.90
2	2	1 2 3	9	8 10 12	485 495 510	250 ms	10 18 45	249.82
	5	1 2 3	10	8 10 12	295 300 395	250 ms	14 25 50	249.75
	6	1 2 3	13	10 12 14	690 850 900 $r_2 = 1645$	250 ms	25 50 μs 75 μs	249.25

k	n	t^2_k	u_{nk}	C^2_{nk}
1	1	249.98	620	4700
	2	249.86	385	5000
	3	249.90	175	5500
		$r_1^* = 7512$		
2	1	249.82	150	1800
	2	249.75	75	2125
	3	249.25	25	22,350
		$r_1 = 7860$		

Figure 4.5. Model Analysis (Continued)

the required value $D_{min\ kj}$. Any sensor having a greater threshold value than the required value is eliminated from consideration. The remaining sensors' maximum measurement values $d_{max\ ji}$ are compared with the maximum measurement value $D_{max\ kj}$ in the same order of decreasing costs. The first sensor having a maximum value greater than the required value is selected as the minimum-cost solution for that instrument type j and configuration k. Next a measurement range is computed for the sensor. Using the specified resolution value R_{kj} for sensor configuration k and sensor type j, the number of bits for the converter is computed. The sensor measurement cycle or delay time is subtracted from the remaining adaptive control response time t_{mk} and is part of the output state vector. In this example, no delay occurs in measurement. The procedure is repeated for all sensor configurations and sensor types. The return from this stage is the sum of all the minimum costs of the sensors selected.

Stage 2 has a vector input with the sensor configuration k, adaptive control time remaining t^1_{kj}, and converter bits r_{kj}. Converters under consideration are arranged by increasing bit size. Both converter costs and conversion time normally increase with increasing bit resolution. Input state bit resolution requirements r_{kj} are compared with the converter resolution bits y_{lj}. The first converter bit resolution which is greater than or equal to the input bit resolution is selected for the sensor type. This selection procedure is performed for every sensor type and configuration. The return for this stage is the sum of selected converter costs.

Conversion time for the selected converter α_{1j} is subtracted from the input state t^1_{kj}. This result is the amount of time remaining for the processor to execute the longest program t^2_k and the output state from stage 2.

Finally, the logic processor is selected at stage 1. The processors are listed by increasing costs. Time remaining in the input state t^2_k is compared to the processor times. In the example, note that processor costs and times are significantly greater for configuration 1. Configuration 1 sensors measure indirect values of cutting force and require a more sophisticated logic program. The first processor time which is less than the input state time is the minimum-cost processor. The returns are summed with the preceding stages for each configuration, and the minimum-cost configuration is identified. In the example problem, configuration 1 is the minimum-cost solution. Hardware components associated with the minimum-cost solution are indicated in Figure 4.5.

Example Problem Postoptimization Analysis

In the example problem, an LVDT was used to measure the feed rate and distance. The LVDT was selected to obtain high accuracy within a short time period. LVDT costs accounted for more than 50 percent of the sensor costs. Since over 100 milliseconds in the adaptive control cycle are not required by the minimum-cost processor, a less expensive optical encoder can be used in place of the LVDT. Encoder costs depend on required measurement resolution and operating speeds. In order to meet the measurement requirements, a parallel pair of encoders must be

used, costing $1100. The counter (converter) costs $75. Thus, the lowest sensor configuration cost for stages 2 and 3 is $1175. The time for one pulse is 78 milliseconds after the count is initiated at the beginning of the control cycle. This measurement time is within the remaining 100 milliseconds of adaptive control cycle time. This results in a lower adaptive control system cost of $7325 for the second configuration.

Configurations can be changed by adding new components and changing processor execution times and costs to reflect the new configuration. Costs for each new component are computed at each stage with respect to the technology requirements. These costs are then added to the returns previously computed for the stage. For example, the high-speed tachometer might be used in addition to the low-speed tachometer in configuration 1. This increases system reliability. Computer time and costs do not change since the same machine is being evaluated. Thus, component type 5 is added to configuration 1; the minimum cost for the new configuration is $7980. This is an example of the postoptimization design analysis which can be performed using the model.

Summary

The adaptive control model is able to analyze many different sensor design configurations. Sensor performance characteristics are described in general terms. By describing sensors and other system components in these terms, new technologies can be evaluated within the model's current structure.

Using a dynamic programming solution procedure, minimum-cost components can be determined for each sensor configuration being considered in the model. The solution procedure is greatly simplified by evaluating the operating characteristics of different hardware components.

Postoptimality analysis for evaluating and improving the system can be performed using the model's initial results. The constraining relationships are identified in the solution. With these relationships, more cost-effective designs can be analyzed without solving the entire model again.

The model has been set up as a multistage/serial system. The solution procedure can be implemented on a computer. Thus, a data base of component performance characteristics and costs can be maintained. These components can be specified during an analysis. The multistage optimization procedure requires that only the state and returns be stored in memory during the solution. Such a computer program can be implemented directly in the initial design phase of adaptive control systems.

As illustrated in the adaptive and hierarchical control model applications, all levels of the manufacturing system can be analyzed using the basic model structure. Each process block represents a specific system component. Selection of a specific component is determined by satisfying operational requirements, based on the input state which results in the minimum cost for the entire system.

An optimal solution is identified by the model analysis. This component configuration is a benchmark for performing simulation analysis. This mathematical approach saves substantial simulation analysis time. In addition, postoptimality analysis can be employed to identify other optimal solutions based on simulation results.

CHAPTER FIVE

SIMULATION MODEL DEVELOPMENT

Mathematical models can be used to determine an initial optimal computer control design for any level of the manufacturing process, based on system/component operating characteristics and cost data. However, manufacturing systems are dynamic and can have extreme variability at the process level. The purpose of the computer control system is to monitor and control this variability as needed to satisfy manufacturing process operational requirements. A computer simulation model is developed to evaluate the dynamic and stochastic characteristics of the initial computer control hardware design and control strategies obtained from the mathematical analysis.

The combined use of both mathematical and simulation modeling provides more flexibility and accuracy in evaluating manufacturing systems, while substantially reducing analysis time. Initially, only one design must be simulated, since the mathematical model has identified an optimal (minimum-cost) design based on all hardware components and control strategies under consideration. In addition to evaluating the dynamic and stochastic behavior of both the process and control system, simulation results can provide validation feedback data to the mathematical model for postoptimality analysis. As discussed in Chapters 2 and 3, postoptimality analysis can identify updated cost-effective designs based on the simulation results, without re-solving the mathematical model.

A simulation modeling structure is developed to evaluate all levels of the manufacturing computer control system from the microprocessor response for an adaptive control machine process to the factory control logic of a mainframe computer. In addition to the control system simulation, the modeling structure also simulates the stochastic and dynamic aspects of the actual manufacturing processes being controlled. This enables "real-time" evaluation of control actions and manufacturing operations. The model structure can be used to evaluate all phases and components of both the manufacturing process and control system simultaneously, from CNC control loop stability to production flow programs for entire factories.

Requirements for the simulation model are presented in this chapter. The choice of SLAM as the simulation language is discussed. Using the network portion of SLAM, the model structure is developed with only three basic system components. An entire manufacturing production cell, including an adaptive control turning operation, is simulated with the model. Several tests and evaluations are performed using this simulated cell to demonstrate the flexibility, simplicity, and comprehensive nature of the simulation structure.

Simulation Model Requirements

Analytical Capabilities

The simulation must be able to analyze the dynamic behavior of computer control system responses to random variations in the manufacturing system being controlled. In order to evaluate the entire manufacturing system, the simulation model must address all the processes and control levels shown in Figure 5.1. The detail of each process or control simulation depends on the component level being designed. For example, if a hierarchical control network is being investigated, general time distributions can be randomly sampled to determine process times. However, an adaptive control system design analysis requires a detailed simulation of machine/workpiece dynamics and variations. All these levels of detail have to be simulated simultaneously for the manufacturing process and control network. The manufacturing process is the primary source of variability in the simulation model. All levels of computer controllers must be able to respond to and control these manufacturing process variations. The computer control system is designed, simulated, and evaluated with respect to the manufacturing process.

Manufacturing processes and control systems have both discrete and continuous events occurring simultaneously. For example, a computer instruction to a servodrive is a discrete event, while the motor response could be a first-order differential response with respect to time. Cutting temperature might be continuous with respect to time; while the sensor monitoring the temperature might be simulated as a

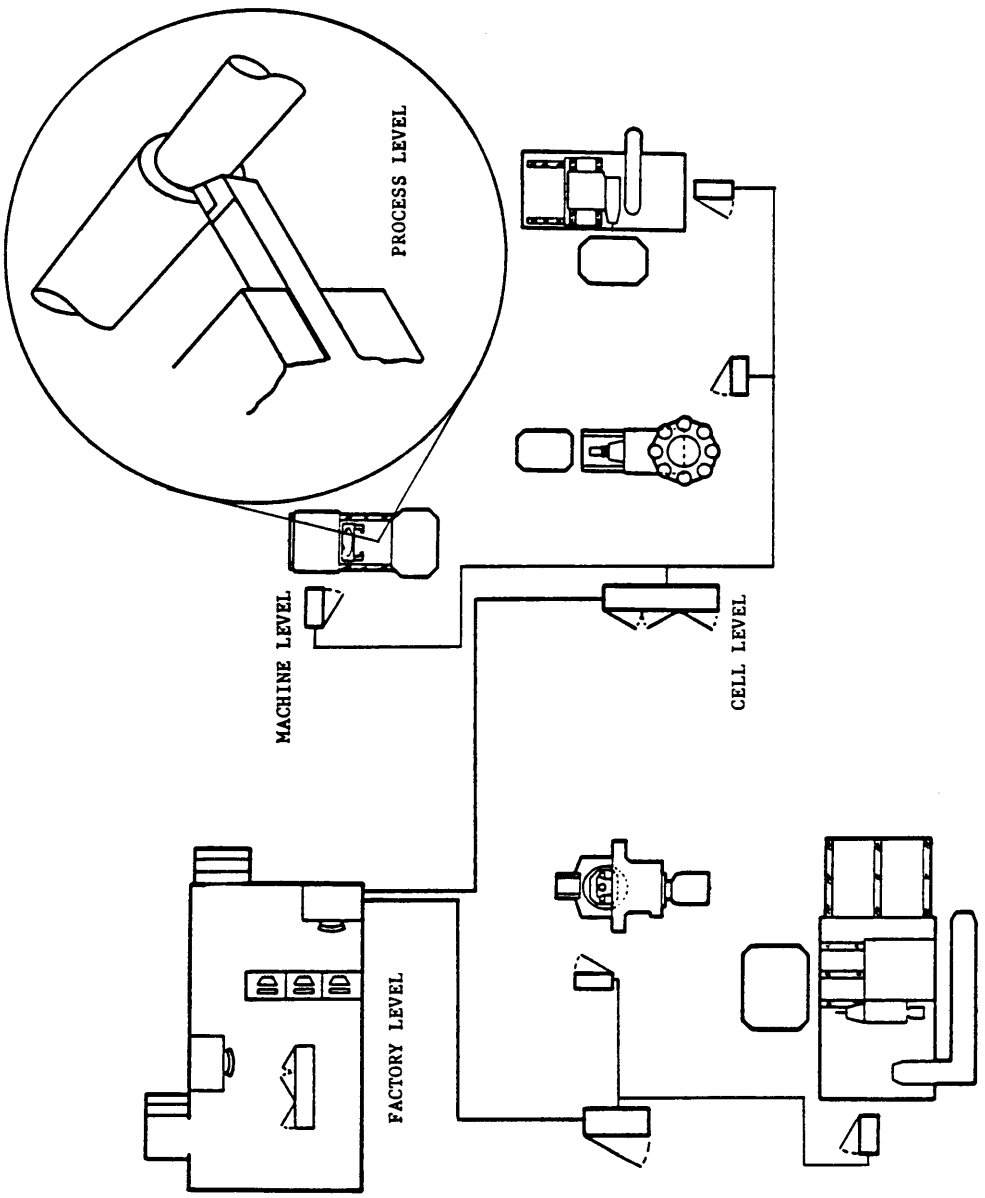

Figure 5.1. Computer Control Levels in CIM

discrete process. For example, if the sensor is triggered at a critical temperature level, a time event occurs when the sensor detects the temperature. Thus, the model must have the flexibility to simulate both discrete and continuous events.

The model must have real-time output, so control actions and responses can be examined together. Actual system control components are evaluated in a continuous time domain. For example, a tool holder strain gage voltage output over time is recorded on a strip chart along one channel, providing a continuous plot of voltage over time. This plot and other machine tool control and output variables might be required to evaluate an adaptive control system's performance. The simulation model must have a similar output format for validation analysis with the actual data. In addition, events leading up to the control action must be known to evaluate control strategies and equipment responses. Output format is important. All information should be available to the designer with respect to a common time base, such as processor cycle time, and similar to real-time output from actual process measurements or design projections.

Software evaluation is an important capability of the simulation model. Software is a costly component of the design process and is usually developed independently of the control system. Performance of the software for the entire factory usually cannot be tested before implementation on the real system. A manufacturing system simulation model capable of analyzing control logic and system responses can greatly reduce software development and evaluation cost. Thus, the

simulation model must have the ability to evaluate software in simulation time. This type of simulation provides manufacturing system dynamic response characteristics to software logic. In addition, all states in the decision matrixes can be identified and tested. Such a simulation model emulates control system responses with respect to variations in the manufacturing processes.

Design and evaluation of manufacturing control systems require that many different details of the system be simulated. For example, CNC part machining time could be sampled from a known distribution if an entire factory were being evaluated. However, an adaptive control system evaluation requires that the dynamics of machine drive motors and tool cutting forces be simulated using continuous functions with respect to time. Therefore, the capability to address various levels of physical detail must also be included in the model's structure.

Modularity is the key to the simulation model and greatly enhances sensitivity and future design analyses. The entire manufacturing process is complex and cannot be accurately developed in a single simulation model structure. Each component of the process and the control system should be developed and tested outside the control network simulation model. Once evaluated, these component and process modules have to be combined in a common network structure. A structured simulation network must have the same "plug-in" capability as the physical system. This capability enables future systems to be incorporated in the model and analyzed with the present system.

Simulation Language

SLAM is selected as the simulation language for performing the computer control system simulation. SLAM meets the simulation analysis requirements and enhances the development and evaluation process. SLAM can simulate discrete and/or continuous events. The network structure of SLAM provides the modular design capability required of the simulation model. This network simulation capability enables the control system to be represented in a structured format.

SLAM is well documented and is not a "black box" simulation language. SLAM is written in FORTRAN IV; therefore, modifications can be made to the actual source program. SLAM performs statistical collection and computation, addressing both time-persistent and observation statistics. The continuous plots generated by SLAM can be used directly in the design process. Actual process control programs are executed in FORTRAN IV and evaluated directly using the discrete-event portion of SLAM. No other simulation language has the flexibility and capability to meet the requirements of the manufacturing control simulation model being developed.

Simulation Model Structure

The simulation model is based on three major modules: the control computer, interface, and process dynamics. All production computer control systems can be represented using these basic modules. The modules enable very complex systems to be broken down into well-defined subsystems. These subsystems are easier to debug during model development and enable a better understanding of the physical system

and model results. A network of modules and their relationship to a manufacturing system is shown in Figure 5.2. Control computers are linked to a process by an interface module. Hierarchical control network computers are connected by I/O interfaces, which include communications. As in the actual physical system, these interface modules enable different types of computers and control equipment to be interconnected. The modeling detail of each module depends on the data available and the scope of the simulation analysis.

The actual process being controlled is also simulated using the SLAM network, event, and continuous simulation capabilities. Each process of the manufacturing system is "plugged into" the control simulation structure with a process interface. Since processes are "plugged into" the control computers, a simulated process responds to commands from the control computer and reacts as modeled. Thus, control software must be developed for each process in the network.

The simulation program structure for each basic module is developed using SLAM. Major input and output requirements are addressed. Basic control system component simulations with each module are presented. Different levels of simulation detail are discussed, as well as SLAM's flexibility for process simulation. Manufacturing process simulation modules are developed for unique control situations.

Control Computer Module

The computer is the center of the control system. Figure 5.3 is the SLAM network representation of a computer module. The EVENT node receives information to be processed by the computer from the network.

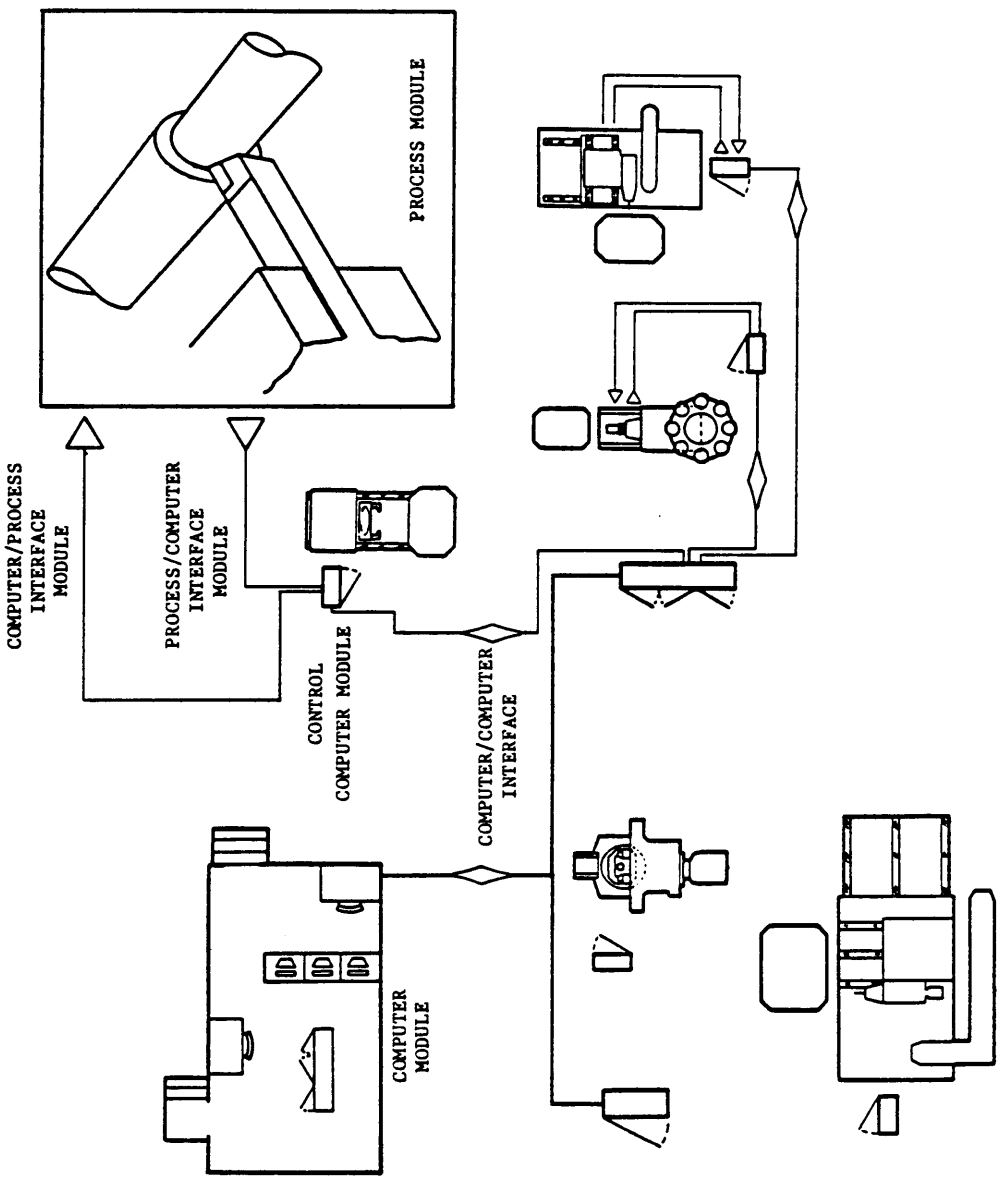

Figure 5.2. Basic Component Modules for CIM

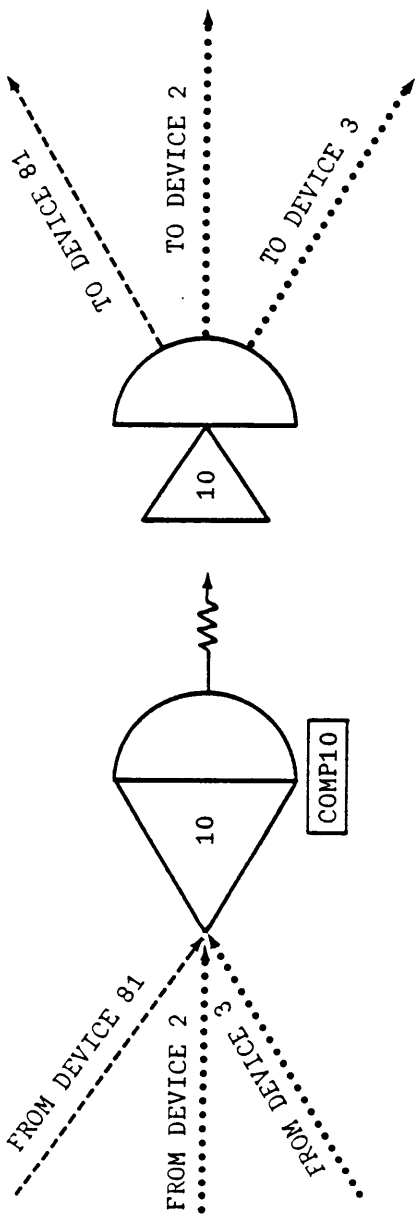

Figure 5.3. Computer Module

The ENTER node sends commands and information from the computer to other computers and devices in the system. When a message comes into the computer through the EVENT node, the computer calls the appropriate software to service the message. The execution time is simulated, and an appropriate message is sent out through the ENTER node to the device determined by the software logic. The computer module is capable of simulating and analyzing the following computer and I/O functions in real time: (1) software, (2) memory and storage, (3) program execution time, and (4) interrupt service routines. The structure and software development procedure are presented in the following sections.

Computer Structure

Each computer is given a node number to identify the computer in the simulation control network. Thus, the EVENT and ENTER nodes have the same number. A convenient numerical identification method begins with the highest level computer of the hierarchy. Computers can be grouped in values of tens (Figure 5.4), leaving unassigned numbers at each level for adding computers and processes in the future. This numbering sequence enables the user to keep track of the computers and simplifies program development.

A computer event occurs when an execution is completed or a message event is sent to the computer module. The SLAM EVENT subroutine carries all the overhead for these events, including interrupts, pooling, and output from the computer module. Every time a computer module event occurs, the EVENT subroutine calls the appropriate software. Subroutine EVENT I/O message structure is

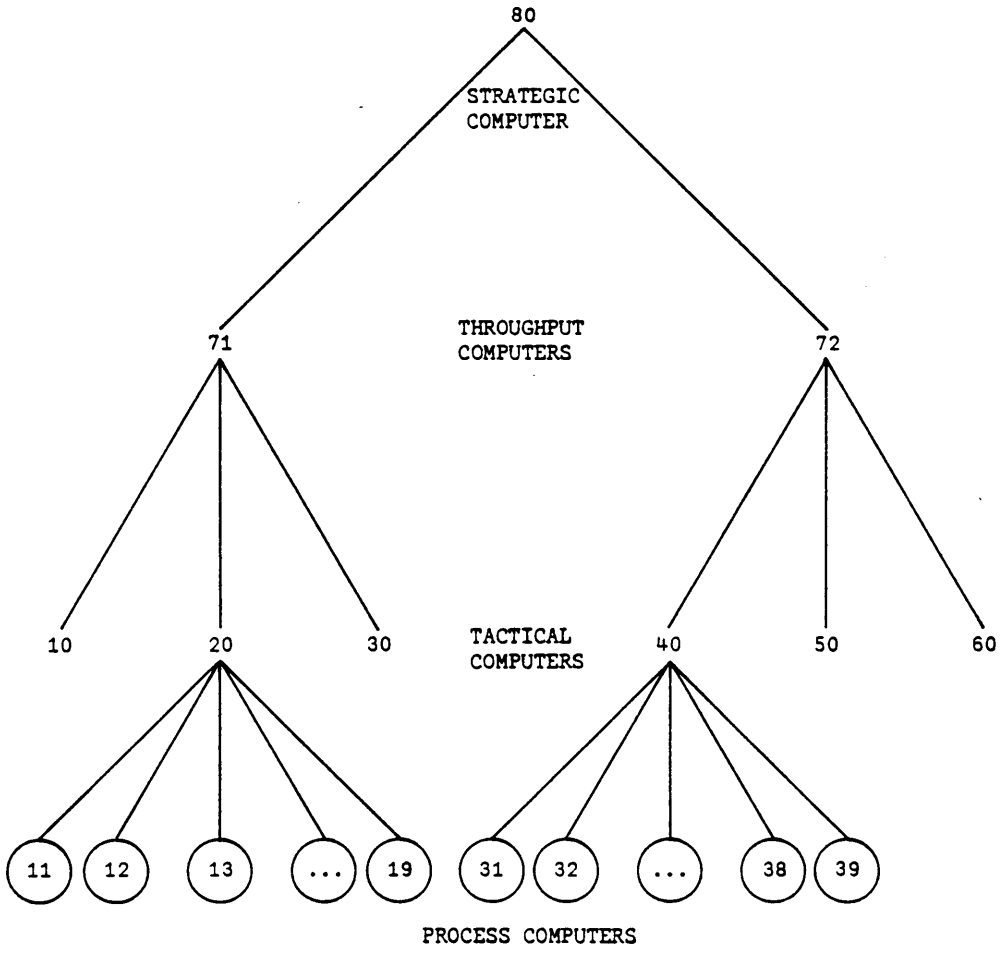

Figure 5.4. Identification Hierarchy for Computer Modules

discussed in the interrupt processing section. A computer event in the network makes a direct call to the SLAM EVENT subroutine by the module's number. Subroutine EVENT in turn calls the appropriate computer service subroutine, based on the current processor conditions of the computer and service request priority.

The actual control software resides in the discrete-event portion of SLAM. The software is written in FORTRAN IV and is called by subroutine COMP**, where ** is the computer identification number. Figure 5.5 is the basic flowchart for implementing the computer service routines. The service subroutines called by the COMP** subroutine are executed according to the device requesting service. Notation for subroutine identification is as follows: C** is the service computer and D** is the device requesting service. In the example shown in Figure 5.5, subroutine C01D10 is called when a message is sent to computer 1 from device 10. Device 10 in this case is the higher level computer.

The message event contains seven control attributes in an array labeled ATRIB. When the event message occurs, the SLAM EVENT subroutine is called and the ATRIB array contains the message event's attributes. ATRIB(3) is the current computer event being requested. ATRIB(2) contains the requesting device number. In the example, subroutine EVENT calls subroutine COMP1 based on ATRIB(3). Subroutine COMP1 calls subroutine C01D10 using ATRIB(2), which contains the requesting device's number.

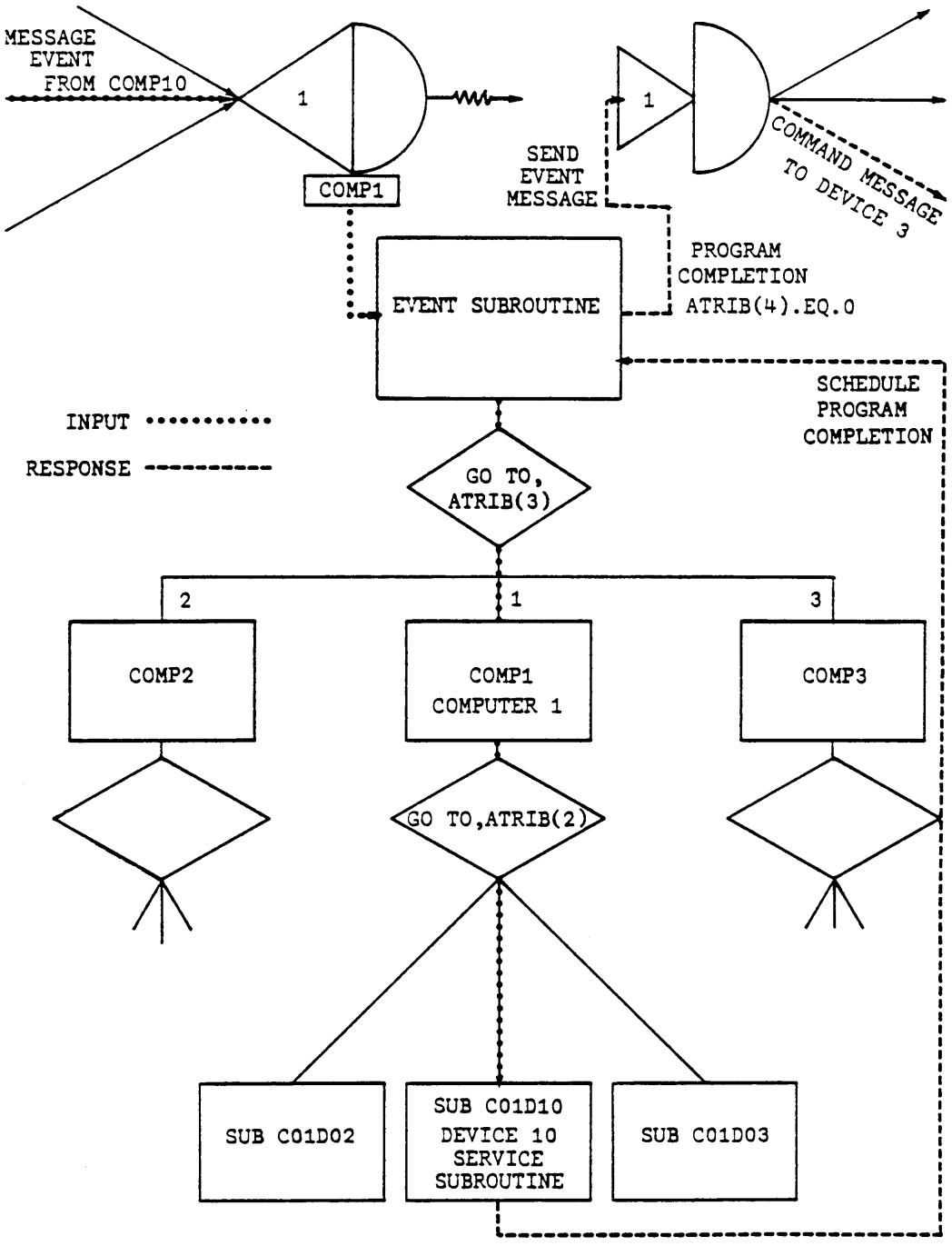

Figure 5.5. Computer Service Subroutine EVENT

The control program is executed. A message event is scheduled for device 3 based on the program logic. ATRIB(2) is set to the current computer identification number 1 and ATRIB(3) to the destination device identification number 3. ATRIB(4) equals zero to indicate that a program has completed execution. A program completion event is scheduled for computer 1. After the program has completed simulation time execution, the computer's program completion event is executed by the SLAM EVENT subroutine, and the message event is released into the network for device 3.

The subroutine EVENT has been designed to handle the overhead of I/O and other computer functions. Only the control service subroutines must be developed. The structure of subroutine EVENT increases simulation program execution speed, while reducing simulation development time.

Software and Execution Time

The software logic is executed in the SLAM discrete-event subroutines. In an actual system, a computer takes a finite time to execute the program sequence. Program execution time varies according to the number of program steps executed. This execution time can be extremely critical when simulating microprocessor control systems requiring fast responses to process variation (e.g., adaptive control systems). Programs with a very long execution time might cause slow control responses to machine variations, creating system instability and accuracy problems.

Simulated execution time is included in the software structure. Depending on the accuracy required, program execution time for each program loop or sequence is summed, starting at the beginning of the program. These times can be absolute values or random variables sampled from known distributions based on I/O, processor, and memory speeds. Once SLAM has finished the execution of the program, completion of the program in actual simulation time (TIME) is scheduled based upon the sum of execution times. SLAM subroutine SCHDL(COMP**,TIME,ATRIB) schedules the program completion event. Subroutine EVENT is called when the simulation program TIME has been completed. ATRIB(4) indicates program completion event status. Subroutine EVENT releases command and message events into the network based on the program logic results.

An example of the execution time accounting system is shown in Figure 5.6. The program checks a sensor input for a trigger value. The program is a loop. Approximately 104 microseconds are required to read the sensor and execute. TIME is summed for each execution of the sensor pooling loop. When the sensor value is finally obtained, the program jumps out of the loop and schedules a program completion time. This simulated program execution time depends on the number of loop executions. The initial time at the beginning of the loop is the amount of time required for the processor to call the service program and return from the subprogram.

With execution time imbedded in the software, the computer module is an emulator of the physical computer. For time-sharing systems, the

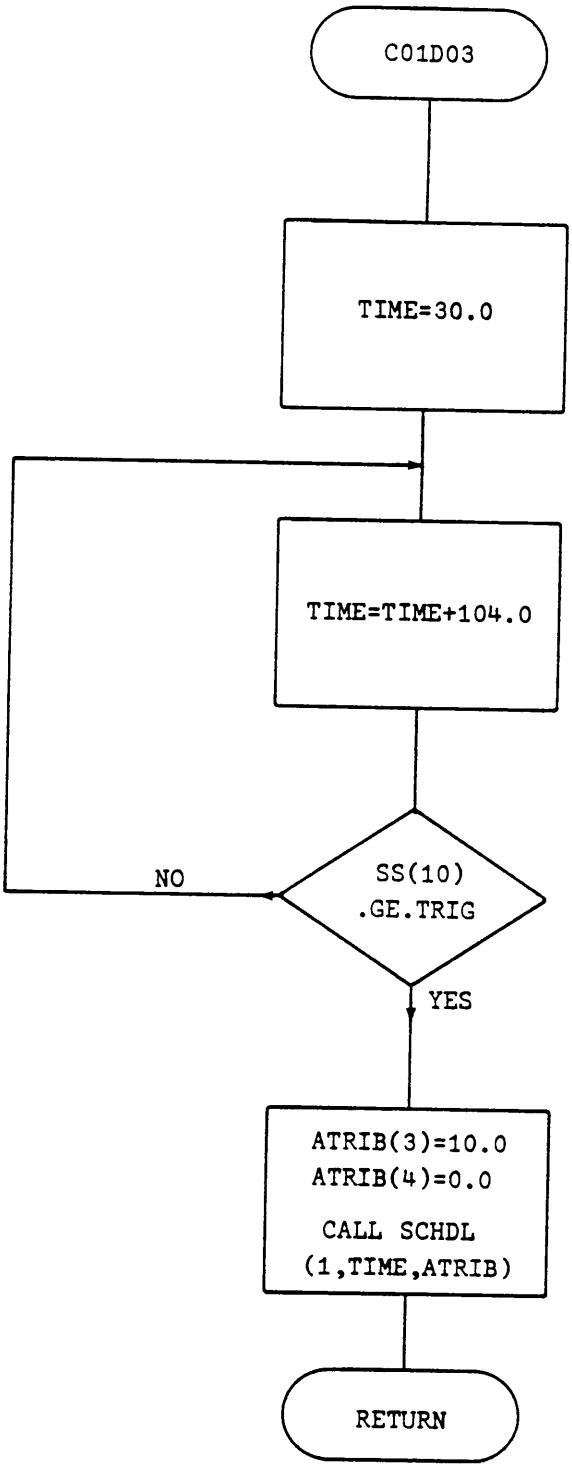

Figure 5.6. Timing of Program Execution

execution times can be factored by a scale based on computer utilization. Thus, with greater computer utilization, processing time for each program increases. The execution speed can also be simulated as a function of memory usage. Again, the model is very flexible. The degree of sophistication depends on the detail and size of the actual manufacturing system or process being investigated. Computer execution time of the factory control system can be insignificant when compared to process and transfer times and might not be considered in the analysis. However, the part scheduling and routing software logic evaluation is very important in such an analysis.

Memory and Storage

The real-time storage of data, service programs, and operating systems is contained in SLAM files. Each CPU's RAM is represented by a file. The size of the memory can be specified by dividing the RAM into the smallest data storage increment possible under the operating system. The number of increments available minus operating system space and I/O buffer is the memory available. The file size is specified by SLAM control statements for each computer node. If the file exceeds its limit, an error message can be processed and sent to the originating device.

In manufacturing systems, the program size usually remains constant over time, but data varies greatly. Therefore, file sizes can all be based on the number of data inputs which can be stored. File entries can be located with SLAM functions, such as NFIND, based on any attributes associated with the entry. Entries can be stored in the

files, either first-in first-out (FIFO) or last-in first-out (LIFO). Entry ranking can also be based on an attribute value using SLAM control commands [58].

Auxiliary files are used as offline part storage devices. Process data can be stored in the SLAM attributes in the auxiliary files. When the information is needed, a search can be made of the auxiliary file. For example, a part program might be stored in the auxiliary file. Each attribute might contain several machining steps for the cutting depth of a lathe over time. Data attributes are discussed in the message protocol section.

Figure 5.7 shows the different memory and storage device configurations. In addition to the SLAM files, process information can be stored as data arrays in the discrete SLAM FORTRAN program. The type of storage configuration or procedure employed depends on the scope of the process being simulated. Any degree of detail can be evaluated, including variations of the basic memory configurations.

Interrupt Processing

One of the unique features of this computer module is the ability to simulate I/O interrupts. Interrupts are processed immediately by the EVENT subroutine according to their priority. An interrupt request arrives at the processor's EVENT node from a device requesting service. The message interrupt priority is contained in the message event ATRIB(1) and is interpreted by the EVENT subroutine logic. If the interrupt's priority is lower than the priority of the program being executed, the interrupt message is stored in the computer's file with

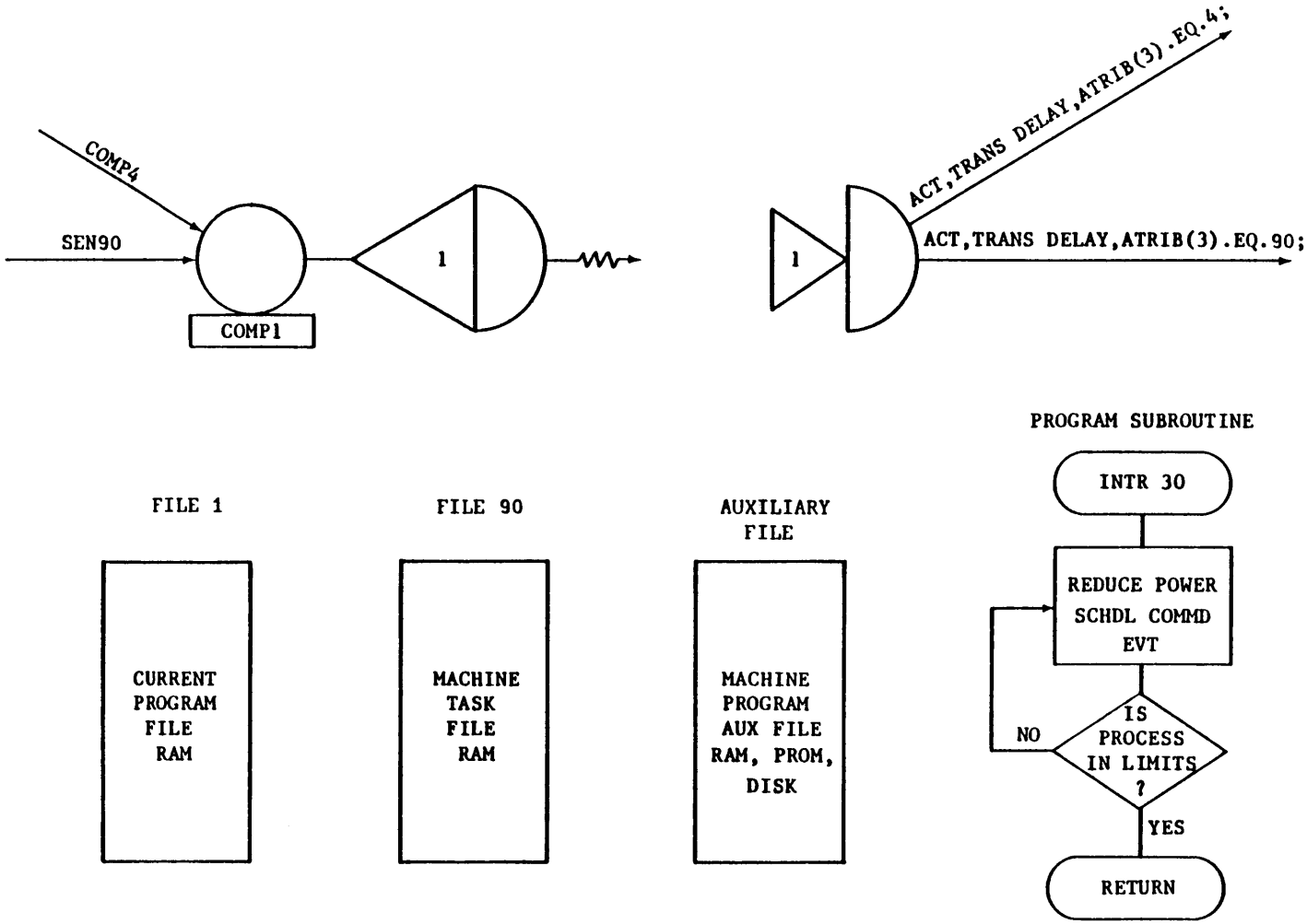

Figure 5.7. Computer Module Memory and Storage Device Configurations

other service requests. Interrupts are stored in the queue according to priorities. When the current process has reached scheduled completion, the next highest interrupt is removed from the file and processed.

When the interrupt has a higher priority than the current service program, the current program is stopped at its execution time. Values are stored in the computer's SLAM file. The computer's interrupt file has the same identification number as the computer's EVENT node. Entities are stored in the file by highest value based on ATRIB(1). The current program is removed from SLAM's event calendar (NCLNR) file. The remaining execution time is recorded. The interrupt request is serviced. When the interrupt program has completed execution, the interrupted program is pulled off the stack and resumes execution.

In addition to interrupt capability, the computer can also pool devices. The pool priorities are lower than the interrupt priority (Figure 5.8). A "break" number is used to distinguish the interrupts from the pool I/O requests in the computer's file. In Figure 5.8, all values of ATRIB(1) greater than or equal to 5.0 are priority interrupts. The pool inputs are arranged in the order in which they are queried by the computer. When the device sends a message, the EVENT subroutine determines if the device is a pool request instead of an interrupt request. If the processor is busy, the service request is stored in the processor's input file, according to the priority, which is the first attribute.

	ATRIB(1)	ATRIB(2)	ATRIB(3)	
INTERRUPT REQUESTS ATRIB(1)≥5	6.0	5.0	1.0	...
POOL REQUESTS ATRIB(1)<5	3.4	10.0	1.0	...
RAM PROGRAM DATA ATRIB(1)<0	-1.0	1.94	2.68	...

Figure 5.8. Computer Module Priority Interrupt Structure

The processor keeps removing requests from the file according to priority. The interrupts are serviced first, then the pooling requests. When a second service request is made by a pooled device which has not been serviced since its last request, its last request is replaced in the file by the current request.

An ATRIB(1) less than zero indicates that the ATRIB array is data stored in RAM. For instance, data for tracking parts might be stored for parts in the cell. This file could be ordered by device position; for example, a part on machine 2 would occupy the second position in the computer's file with a negative ATRIB(1). Other information can be stored with negative ATRIB(1).

All of the interrupt and pooling routines are handled in the EVENT subroutine. Once the priority levels are defined for the system, the EVENT subroutine automatically handles the overhead for filing and removing service requests. This capability enables complex interrupt programs to be evaluated. Routines which are "locked out" from service can be identified. Pooling and interrupt priorities can be changed in the computer's discrete programs. The priority levels for the actual system can be designated from the simulation results.

I/O Interface Module

The I/O interface module connects the computer and process modules together in the control system. This module simulates all types of interfaces and communication systems from a pair of twisted wires to a wideband optical communication system with complex coding and decoding processes. The basic module can be modified to model a wide variety of

local area communication networks, including IEEE-488, Ethernet, and token passing. Communication timing and phasing, multiplexing, pulse code modulation and other signal modification processing techniques can be simulated. Even error checking codes can be simulated with the communication link in conjunction with the processor and computer modules. In addition to physical equipment and processes, message protocols are also simulated and evaluated.

The basic SLAM I/O interface module is an ACTIVITY branch and ASSIGN node. Two computer modules are connected together with two ACTIVITY branches for two-way communications. Before discussing the I/O interface module, the message event control attributes are presented, including the information/data structure. The I/O interface module which "carries" the message is developed. The I/O interface module is modified to simulate local area networks and communication processes, such as duplexing.

Message Protocol

Each event in SLAM has a specific number of attributes stored in the ATRIB array. The number of attributes is specified by the LIMITS statement in the SLAM control input statements [58]. When an event occurs, these attributes are removed from the event calendar and stored in the ATRIB array. For example, when a processor event occurs at the current time TNOW, the attributes associated with the ACTIVITY branch are stored in the ATRIB array. These attributes contain the communication control information and message protocol for the computer control network.

The basic attribute array is presented in Figure 5.9. The first seven attributes contain control information for each message event. These control attributes are used by the computer and process interface module to determine the status of the message. The remaining attributes (8 through 100) can be used for data and information transfer. The attribute message array is discussed below:

ATRIB(1) - (I/O Priority)	I/O interrupt or pooled priority of message event.
ATRIB(2) - (Origin Device)	Device initiating message event.
ATRIB(3) - (Destination Device)	Device receiving message event.
ATRIB(4) - (Process Time)	ATRIB(4)=0.0 - program completion event. ATRIB(4)>0.0 - estimated process time used by the process module.
ATRIB(5) - (Message Length)	Number of data attributes in the message protocol; used by I/O interface module to determine communication time.
ATRIB(6) - (Mark Attribute)	Accumulated message event time in system; used to determine processing throughput and other statistical data.
ATRIB(7) - (BLANK)	Left for future use.
ATRIB(8) through ATRIB(100) - (User defined)	Information transferred from device to device.

All message event attributes can be changed in the discrete or network portion of SLAM.

In the example attribute array, the message data protocol begins with ATRIB(8). ATRIB(8) is the part number associated with the message. ATRIB(9) is the part's manufacturing priority. This priority differs from the I/O priority ATRIB(1).

CONTROL ATTRIBUTES

I/O PRIORITY 1	ORIGIN DEVICE 2	DEST. DEVICE 3	PROCESS TIME 4	MESSAGE LENGTH 5	MARK ATRIB. 6	BLANK ATRIB. 7

PART NO. 8	PART PRIORITY 9	TASK POINTER 10	TASK 1 11	TASK 2 12	...	BLANK ATRIB. 100

MESSAGE DATA ATTRIBUTES

Figure 5.9. Simulation Message Protocol Array

Part machining tasks are carried with the part through the process (Figure 5.9). The tasks start with ATRIB(11). As the part tasks are completed, the task pointer ATRIB(10) is incremented to the next task. The part is assigned to the next operation based on the current production system status determined by the control computer logic.

This example message coincides with a part being simulated in the manufacturing system. If the control logic is faulty, the part will be processed incorrectly and indicated in the simulation output trace. Thus, message protocol and program logic are both evaluated by the simulation structure.

I/O Interface Structure

Now that the message protocol has been presented, the I/O interface module is developed. A network of three computers is shown in Figure 5.10. The I/O interface modules between the higher level computer 10 and the two lower level computers 2 and 3 are SLAM ASSIGN nodes and ACTIVITY branches. The ACTIVITY branches represent communication traffic flow in one direction. Thus, two-way communication requires two I/O interface modules, as shown in Figure 5.10. The top computer sends message events from the ENTER node. Lower level computers receive the message event through the EVENT node.

A computer releases a message into the network at the scheduled control program completion time. An ASSIGN node computes the message transmission time for the ACTIVITY branch. At the ASSIGN node, message

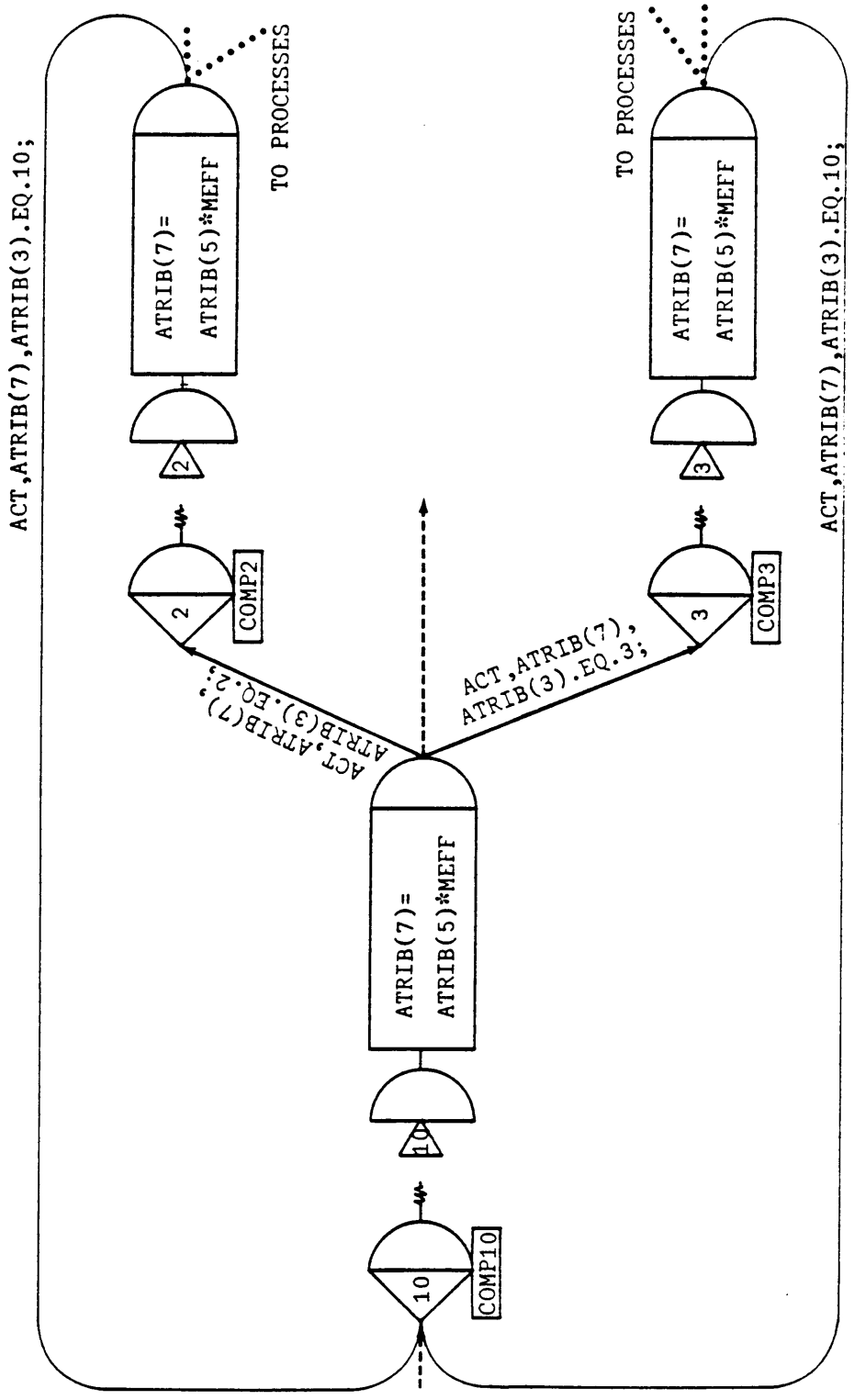

Figure 5.10. I/O Interface Modules in a Full-Duplex Communication Network

length (in bits) is multiplied by the communication efficiency rate. This rate is calculated by:

$$\text{Communication efficiency} = \frac{\text{BITS/CHARACTER SENT}}{\text{BITS/CHARACTER} * \text{BAUD RATE}}$$

BITS/CHARACTER SENT includes the character bits, as well as stop, framing, parity, and other digital control and error-checking communication bits. Communication time for the activity can also be a random variable. The ACTIVITY branch uses this value to simulate actual communication time.

The ACTIVITY branch following the ASSIGN node is determined by the destination device ATRIB(3). As soon as the activity completes the time, the EVENT node "receives" the message with the message event attributes.

When this computer event occurs, the message event is processed according to the processor's current state, as described in the computer module section. If the initial message is a service request, then the computer module sends a ready message back to the requesting computer. An entire message, including system data required for control decisions, is sent to the destination computer through the I/O interface module. Handshaking routines can be implemented and tested. Note, the entire message is stored in the queue when the computer module is busy. ASSIGN nodes can be eliminated if all the communication rates are the same. In this case, ATRIB(5) contains the communication time instead of the message length. However, it is more efficient to use the ASSIGN node for messages and multispeed

communications links, thereby reducing programming overhead on the computer module.

Communication Techniques

The I/O interface modules presented thus far have represented full-duplex communication systems. Computers are able to send and receive messages simultaneously. Half-duplex communications can also be simulated with the I/O interface module. Figure 5.11 shows a half-duplex system between computers 1 and 2 using the SLAM resource AWAIT and FREE nodes. The AWAIT node is placed just before the ACTIVITY branch. When the message event goes through the ASSIGN node, the AWAIT node is encountered. If the line (resource HF1) for the I/O interface module is available, the message event proceeds to the appropriate ACTIVITY branch based on ATRIB(3). If the other computer module has a message to send over the same I/O interface module, the message is filed in AWAIT node because the line (resource HF1) is not available. When the message event has been completed, the line (resource HF1) is freed for the receiving device to respond. In Figure 5.11, COMP10 sends a message to COMP2. If the resource is free, the message goes immediately through the AWAIT node 82 and begins communication processing at the ACTIVITY branch. If COMP2 has a message event for COMP10, the message is stored in AWAIT node's file 82 because the line (resource HF1) is being used by COMP10.

When COMP10's message event is processed, the resource HF1 is freed by the FREE node. COMP2's message event is released from the

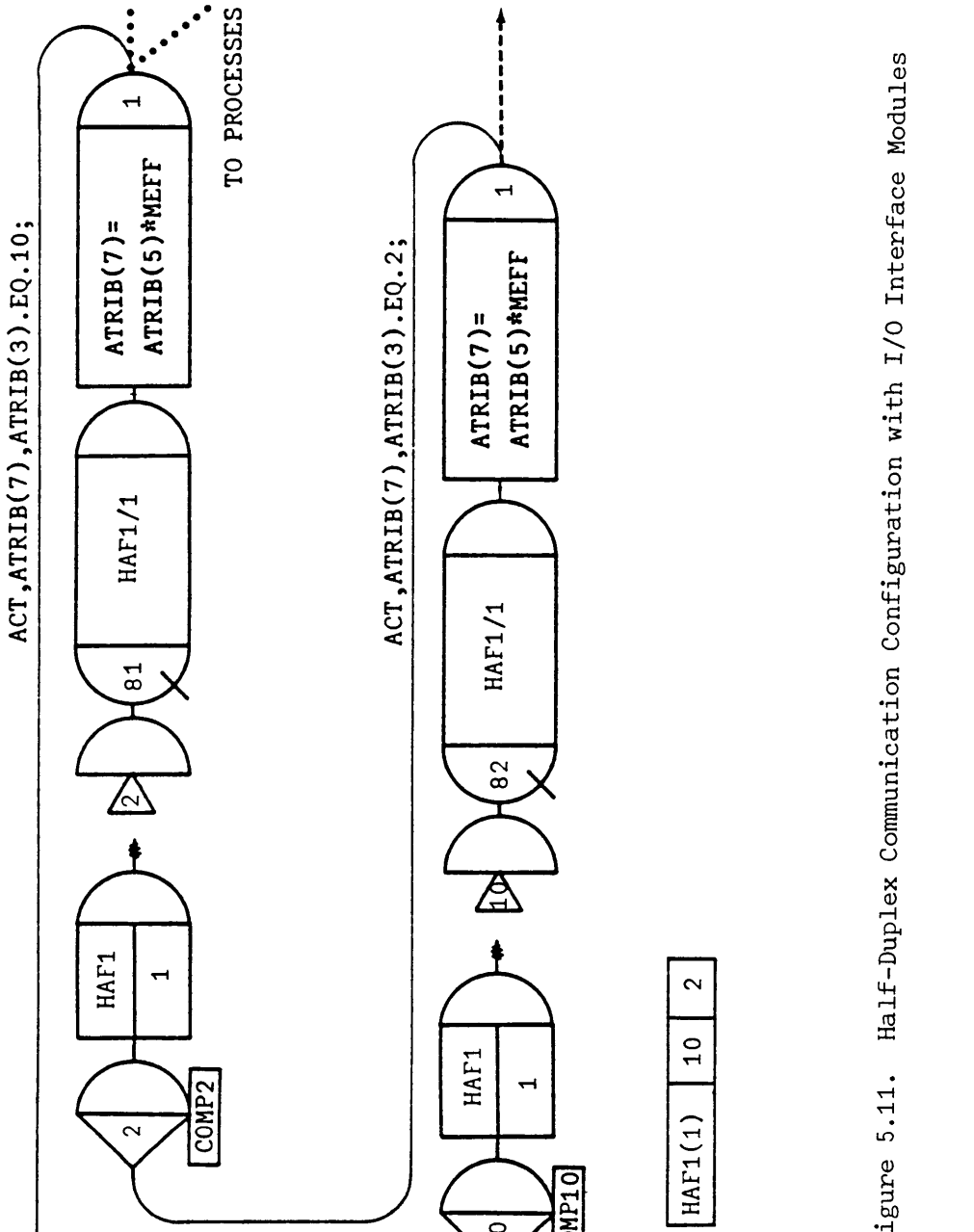

Figure 5.11. Half-Duplex Communication Configuration with I/O Interface Modules

AWAIT node for COMP10. Thus, a half-duplex system is simulated. A separate resource must be assigned to each two-way communication link.

An IEEE-488 network can also be simulated using the resource method. All links share the same resource. The FREE nodes are eliminated from the I/O interface module in Figure 5.12. When a computer module has a resource (NET), all other computers are "listeners" and receive messages from the computer with the resource. Through program logic in the discrete portion of SLAM, the computer can free the resource to another computer and become a listener. Using a similar method, a token passing network can be simulated. (Reference Figure 5.12). The resource represents the token, which is passed from device to device. By ordering the resources in the RESOURCE BLOCK, the token is passed in a specific sequence. In Figure 5.12, the sequence is COMP10, COMP2, and COMP3.

The I/O interface modules can be added to the general simulation network in any degree of detail. Special attention must be given to resource AWAIT and FREE node numbering systems. The file numbers should not be the same as computer module files. Multichannel communication links can also be simulated by setting the number of servers in the ACTIVITY branch equal to the number of channels.

Process Interface Module

The manufacturing process is the most important element of the simulation structure. Process interface modules link the control system with the actual manufacturing process. All manufacturing processes can be simulated with SLAM. The detail of process simulation

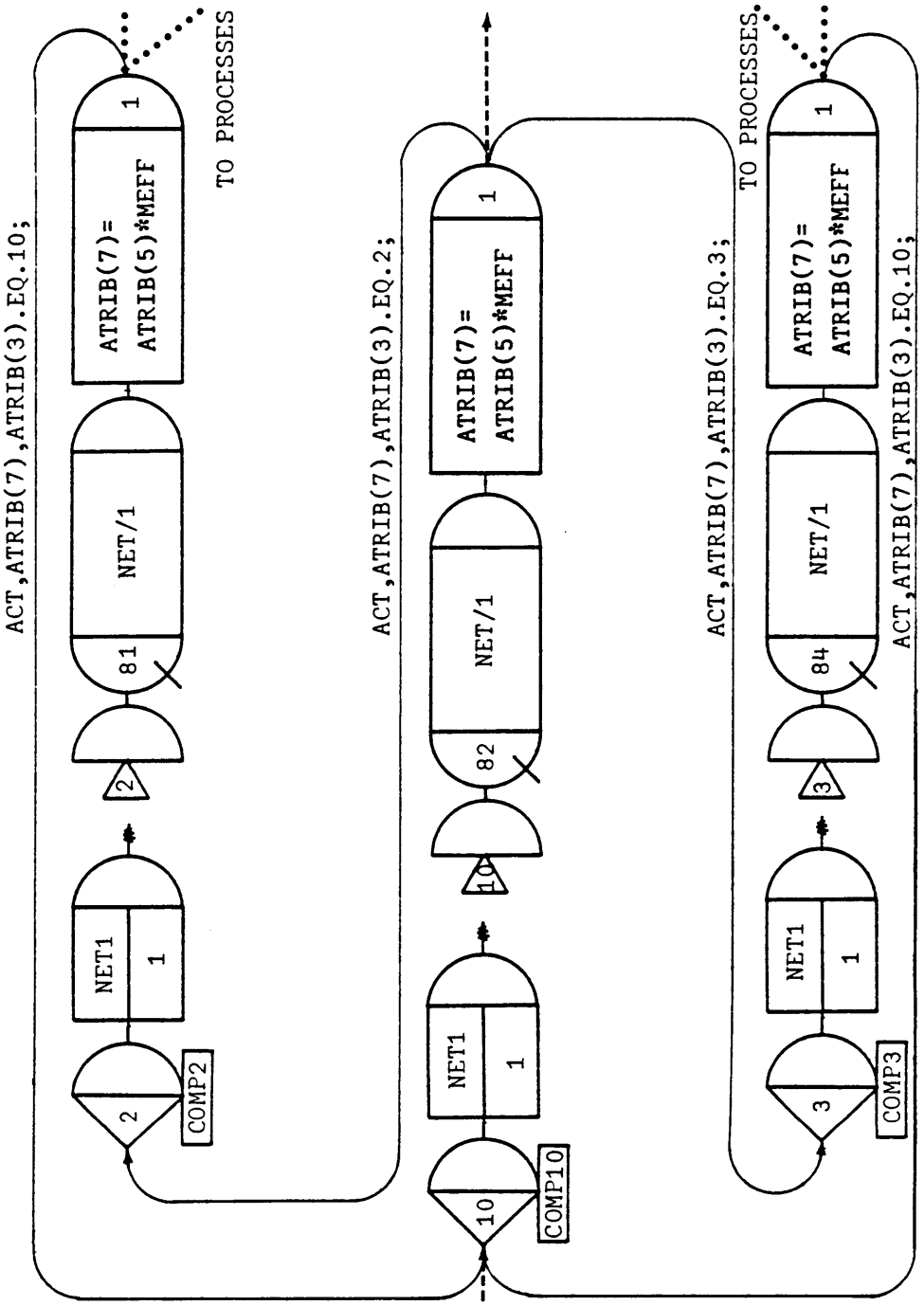

Figure 5.12. IEEE-488 Communication Network Configuration Using I/O Interface Modules

depends on the degree of abstraction. SLAM enables both discrete and/or continuous simulation modeling of a process. For example, the machining time for a part on a CNC lathe could be simulated as a discrete model. When the control computer sends a command to start machining the part, a completion time is sampled from a known distribution. An event message is returned to the control computer when the sampled part machining has elapsed. If machine dynamics (e.g., cutting force, depth of cut) are being investigated, a part would be processed over time according to the dynamic equations describing the turning operation. At the completion of the cutting operation, the continuous model has a discrete event that sends a completion message to the control computer. Processes are simulated to respond to control messages. The process interface module enables control computers to communicate with the processes, just as in a physical system. The control computer must have the correct logic to control the processes.

A basic process interface module is presented in the following sections. The discrete-event processes are discussed with basic network symbols. Continuous process models are also developed. SLAM can be used to simulate the different processes simultaneously. Process modules are "plugged into" the manufacturing system models where control actuators or sensors are located. Thus, both manufacturing and control systems are simulated together.

Discrete Processes

A process interface module can be integrated directly into a SLAM network model of a production system at control and sensor points. The modules consist of SLAM ASSIGN, ACTIVITY, AWAIT, and GOON functions.

Three events usually occur in a manufacturing system control network: (1) control network command to interface module, (2) process response to command, and (3) system status and sensor information to control computer. The commands and process status are transferred through the I/O interface modules.

An example of two interface modules is presented in Figure 5.13. Computer COMP1 controls transfer and CNC machining operations. ACTIVITY branch 1 simulates the conveyor system to the first process PR1. When the conveyor has delivered the part to local storage (AWAIT), a message is sent through the process interface module. This module consists of a GOON node, ACTIVITY branch, and ASSIGN node. A message is sent to the processor at the same time the part (entity) enters the AWAIT node. The ASSIGN node changes or "conditions" the control origin, destination, and priority attributes in the message event. The computer module schedules the machining operation based on the part task and machine.

If the machine ACTIVITY/2 is idle, the computer releases a message event for the FREE node (PR1E). When the message event is received at the FREE node, the actual part entity is released from the AWAIT node. The machining (ACTIVITY/2) branch schedules a completion time. The resource PR1 is again taken by COMP1.

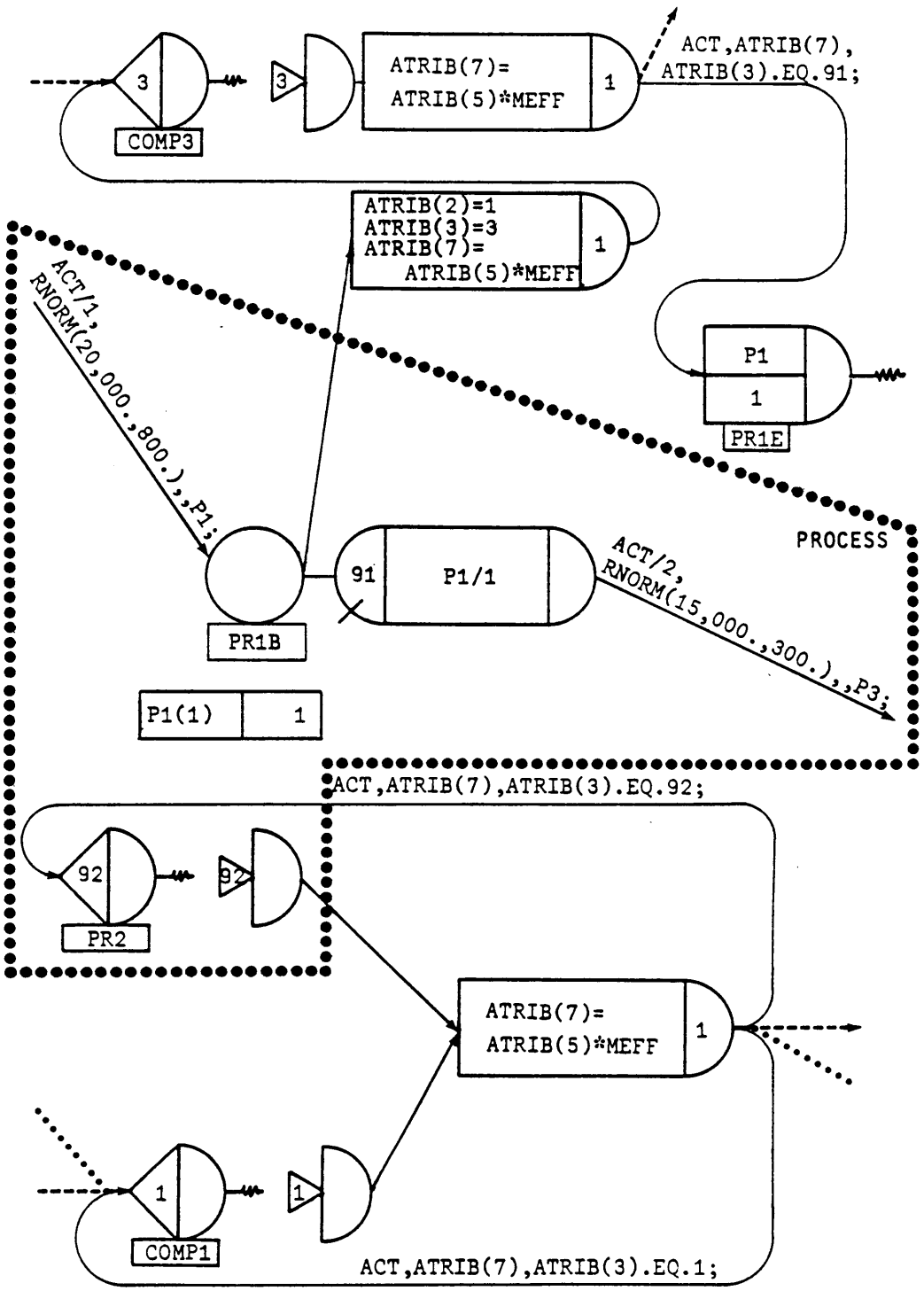

Figure 5.13. Discrete-Event Process Interface Module Integrated Into a Network Simulated Process

A machining operation could be simulated using the discrete portion of SLAM. In this case, an EVENT and ENTER node combination is the process PR2 interface module (Figure 5.13). A part enters the EVENT node, causing the process event to occur. Message event control attributes are specified by the ASSIGN node and sent directly to the control computer COMP2. Logic in the control computer sends message event commands to the process ENTER node. The message event is interpreted by the discrete SLAM logic for the process event. Both the part and process status messages are released through the ENTER node.

Note that these interface modules are the same as the computer modules. Their identification numbers should begin with 100. This numbering system prevents the two modules from being misinterpreted by the SLAM EVENT subroutine. Also, if the control computer cannot identify the part or has a faulty control program, the system will not transfer the part correctly. Other types of process interface module configurations can be developed.

Continuous Process

SLAM can also simulate continuous systems in conjunction with discrete and network models. Before the process interface module for a continuous system is presented, a brief description of SLAM continuous simulation system modeling is discussed. A more detailed explanation is given in reference [58].

A continuous model describes the behavior of a physical system by a set of time-dependent equations. These equations can be algebraic,

differential, and can have stochastic elements. In addition, instantaneous events can occur, directly affecting system status.

SLAM uses state variables SS(I) to describe the current value of time-dependent variables at the current simulation time. The derivative value of the state variable SS(I) is DD(I). An example of a differential equation for a spring-mass-damper system is given in the continuous model block of Figure 5.14. SLAM uses the Runge-Kutta-Fehlberg numerical integration algorithm to determine the state and state derivative variables over simulated time. Algebraic and difference rate equations can also be used when rate changes are constant over time. Thus, continuous processes such as fluid transfer can be simulated.

Process dynamics are also simulated if equations can be developed to describe the system states as a function of time (e.g., for a spring-mass-damper system). Transfer functions for complex systems can be simulated. LaPlace transforms have to be transformed back into their original domain. A combination of continuous and discrete-event simulation enables evaluation of analog systems controlled by digital computers, including a wide range of sensor inputs.

A continuous process control computer receives status data and sends commands at discrete-event times. The process interface module consists of SLAM EVENT, ASSIGN, and DETECT nodes (Figure 5.14). The DETECT node is an interrupt sensor. When a state variable crosses a specified threshold value within a given tolerance, an event is created. The ASSIGN node conditions the signal by assigning values to

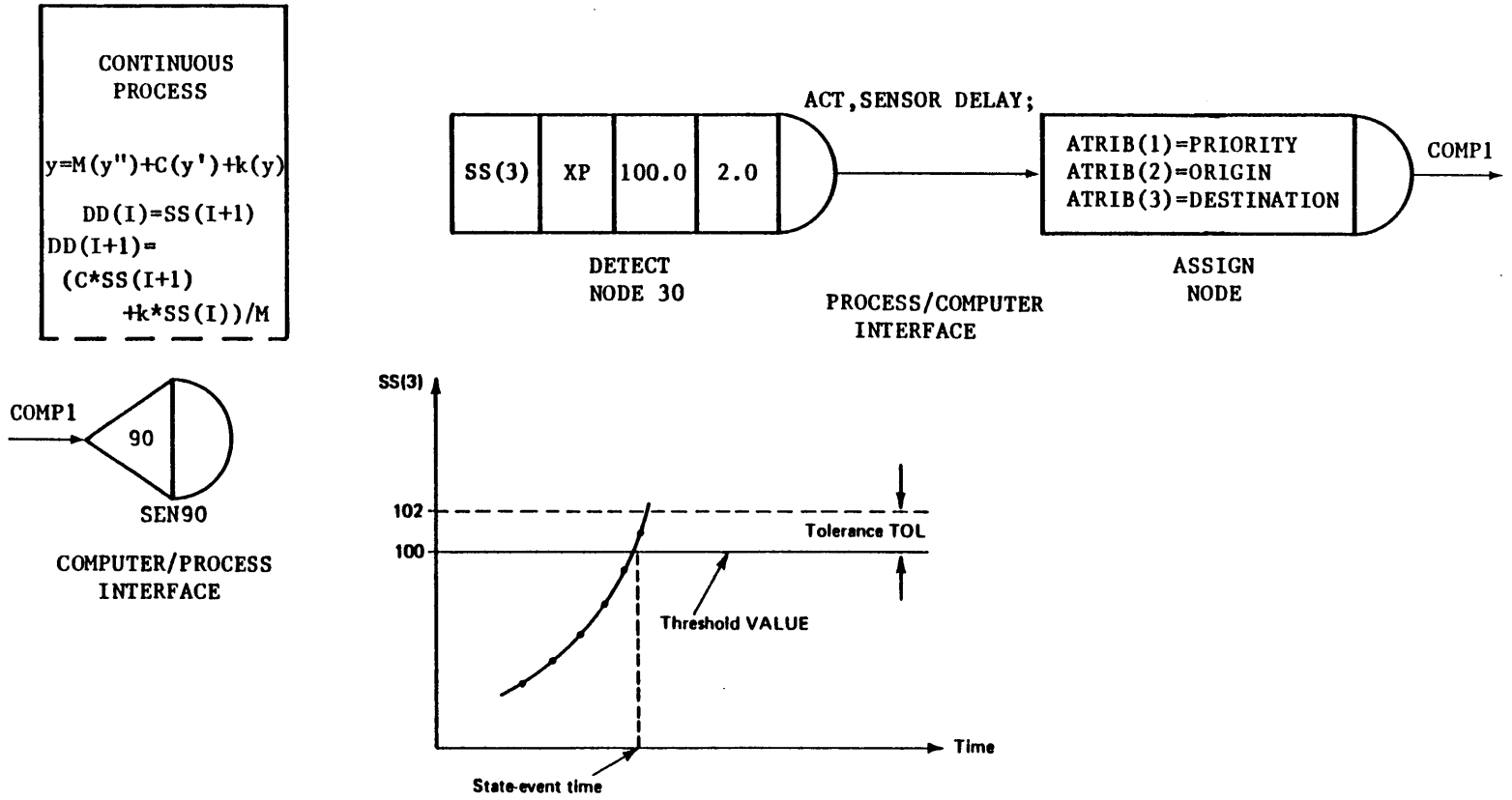

Figure 5.14. Continuous Process Interface Module

the control attributes. The message event is sent to the control computer. A delay value for signal processing can be assigned to an ACTIVITY branch. Thus, message events are delayed before reaching the control computer (Figure 5.14).

Pooling of sensors can also be simulated. A pooling routine is scheduled in the computer program logic. Each state variable at a specific event time can be investigated. Signal conditioning and sensor data conversion delay can be incorporated in the program.

Once the computer module receives the sensor data, a program is executed. A command message event is sent to the processor EVENT node. The program execution and signal conditioning delays are scheduled before the EVENT node is executed. The continuous node EVENT routine is called and the appropriate state variables are instantaneously changed. Thus, control logic, system response, and timing delays can be analyzed over a continuous time period. Detailed levels of the manufacturing process can be analyzed simultaneously with continuous modeling. Thus, small details at the process level, such as adaptive control, can be related to the overall manufacturing system's operation and control.

Model Output and Analysis

Output from the model structure provides both real-time and statistical data over the entire simulation period. Both the manufacturing and control systems can be analyzed simultaneously. SLAM stores statistical data for each module directly in the control network. Besides discrete, network, and continuous simulation of the

statistical data, the SLAM trace option enables real-time analysis of the entire system status at specified time intervals. This real-time analysis capability enables a direct step-by-step comparison of the model and an actual operating system.

The three major types of SLAM data output are: (1) TRACE, system states; (2) STATISTICS, observation and time persistent; and (3) CONTINUOUS, graphical information. All levels of the manufacturing system can be analyzed with these data. Programs can be developed, debugged, and tested. Adaptive control system performance characteristics can be directly compared and validated with actual strip-chart measurements. This wide range of output information provides data for designing, developing, and evaluating control hardware and logic for manufacturing systems.

Trace Data

Computer logic programs for manufacturing systems are generally based on a state decision logic. For example, a part in the manufacturing cell is routed to a specific machine, given the state of all machines in the cell. If all machines are busy, the part must wait. The program logic makes the decision based on the state of the machines and material handling devices. When a part is completed, a state event occurs, and the program must determine the completed part's next machine station, transfer of the part waiting for entry into the cell, and material handling device actuation sequences to transfer the part.

Program logic for performing all the operations is very difficult to develop and test. SLAM trace output provides a "snapshot" of the system states at events and/or specified time intervals. Even the computer memory can be shown at each event output. Output format is determined by the user or the standard SLAM trace output can be used.

An example of a trace output for state status is presented in Figure 5.15. The first machine is the transfer device into the cell. As parts enter the cell, they are transferred by a stepping table to each machine. Each machine has a transfer device. Part programs are stored in machine computer memories capable of performing the current part task. A sequence of transfers and machining operations is present. This information enables software to be tested and debugged before being implemented on an actual system.

Statistical Data

The simulation model can be validated with the physical system and performance determined using statistical data. SLAM provides statistical data based on observations and time-persistent variables. Statistics based on observation are independent of time (e.g., the number of parts processed by a machine). Machine utilization over time is a time-persistent statistic. Both the computer status random variables and message throughput rate are collected in subroutine EVENT. These two statistical outputs enhance the statistical analysis of system performance and validation.

Each module provides statistical output. Figure 5.16 is an example of the SLAM-generated summary report. The computer module

```
 STATE DURING ROUTINE 70; TIME =       451.

 PARTS ON MACHINE       5    2    0    1    0    0    0
 MACHINE STATUS         0    1    0    1    0    0    0
 TRANSFER STATUS        0    0    0    0    0    0    1
 TABLE PARTS            0    4    3    0    0    0

PARTS IN THE MEMORY QUEUES
                        5    0    0    0    3    0    0
                        0    0    0    0    4    0    0
                        0    0    0    0    0    0    0
                        0    0    0    0    0    0    0
                        0    0    0    0    0    0    0
 MACHINE SERVICE TIME   -1.0000 PART   1 TIME=   450.9275
 STATE DURING ROUTINE 71; TIME =       451.

 PARTS ON MACHINE       0    2    0    1    0    0    0
 MACHINE STATUS         0    1    0    1    0    0    0
 TRANSFER STATUS        1    0    0    0    0    0    0
 TABLE PARTS            5    0    4    3    0    0

PARTS IN THE MEMORY QUEUES
                        0    0    0    0    3    0    0
                        0    0    0    0    4    0    0
                        0    0    0    0    0    0    0
                        0    0    0    0    0    0    0
                        0    0    0    0    0    0    0
 STATE DURING ROUTINE  1; TIME =       478.

 PARTS ON MACHINE       0    2    0    1    0    0    0
 MACHINE STATUS         0    1    0    1    0    0    0
 TRANSFER STATUS        0    0    0    0    0    0    0
 TABLE PARTS            5    0    4    3    0    0

PARTS IN THE MEMORY QUEUES
                        0    0    0    0    3    0    0
                        0    0    0    0    4    0    0
                        0    0    0    0    0    0    0
                        0    0    0    0    0    0    0
                        0    0    0    0    0    0    0
 STATE DURING ROUTINE 70; TIME =       489.

 PARTS ON MACHINE       0    2    0    1    0    0    0
 MACHINE STATUS         0    1    0    1    0    0    0
 TRANSFER STATUS        0    0    0    0    0    0    1
 TABLE PARTS            5    0    4    3    0    0

PARTS IN THE MEMORY QUEUES
                        0    5    5    0    3    0    0
                        0    0    0    0    4    0    0
                        0    0    0    0    0    0    0
                        0    0    0    0    0    0    0
                        0    0    0    0    0    0    0
 MACHINE SERVICE TIME  -50.0002 PART   5 TIME=   488.8616
 STATE DURING ROUTINE 71; TIME =       489.

 PARTS ON MACHINE       0    2    0    1    3    0    0
 MACHINE STATUS         0    1    0    1    0    0    0
 TRANSFER STATUS        0    0    0    0    1    0    0
 TABLE PARTS            0    5    0    4    0    0

PARTS IN THE MEMORY QUEUES
                        0    5    5    0    4    0    0
                        0    0    0    0    0    0    0
                        0    0    0    0    0    0    0
                        0    0    0    0    0    0    0
                        0    0    0    0    0    0    0
```

MACHINING OPERATIONS

TABLE TRANSFER OPERATION

TRANSFER OPERATION

Figure 5.15. Manufacturing Cell Trace Output

S L A M S U M M A R Y R E P O R T

SIMULATION PROJECT FIFO: 3 MACHINES BY SCOTT

DATE 11/17/1982 RUN NUMBER 1 OF 1

CURRENT TIME 0.2280E+05
STATISTICAL ARRAYS CLEARED AT TIME 0.0

STATISTICS FOR VARIABLES BASED ON OBSERVATION

	MEAN VALUE	STANDARD DEVIATION	COEFF. OF VARIATION	MINIMUM VALUE	MAXIMUM VALUE	NUMBER OF OBSERVATIONS
TIME IN CELL	0.1231E+04	0.9363E+03	0.7605E+00	0.1018E+03	0.4132E+04	61
TIME BET DEPT	0.3631E+03	0.3695E+03	0.1018E+01	0.1576E+02	0.1561E+04	60

STATISTICS FOR TIME-PERSISTENT VARIABLES

	MEAN VALUE	STANDARD DEVIATION	MINIMUM VALUE	MAXIMUM VALUE	TIME INTERVAL	CURRENT VALUE
CP10 UTIL	0.2455E-02	0.4949E-01	0.0	0.1000E+01	0.2280E+05	0.0
CP80 UTIL	0.4344E-04	0.6591E-02	0.0	0.1000E+01	0.2280E+05	0.0 COMPUTER UTILIZATION

FILE STATISTICS

FILE NUMBER	ASSOCIATED NODE TYPE	AVERAGE LENGTH	STANDARD DEVIATION	MAXIMUM LENGTH	CURRENT LENGTH	AVERAGE WAITING TIME	
1		0.0432	0.2118	2	0	26.6002	
2		0.3303	0.7272	4	0	90.7213	
3		0.1198	0.3321	2	0	70.0637	
4		0.1687	0.4006	2	2	120.1939	MACHINE MEMORY
5		1.2539	1.4553	5	1	402.6685	QUEUES DATA
6		0.1134	0.3342	2	0	42.3755	
7		0.0384	0.1998	2	0	12.1740	
8		0.0	0.0	0	0	0.0	
9		0.0	0.0	0	0	0.0	
10		0.0000	0.0021	1	0	0.0326	PROCESSOR
11		0.0	0.0	0	0	0.0	INTERRUPT
12		0.0	0.0	0	0	0.0	DELAY DATA
13		0.0	0.0	0	0	0.0	
14		0.0	0.0	0	0	0.0	
15		0.0	0.0	0	0	0.0	
16		0.0	0.0	0	0	0.0	
17		0.0	0.0	0	0	0.0	
18		0.0	0.0	0	0	0.0	
19		0.0	0.0	0	0	0.0	
20		0.0	0.0	0	0	0.0	
21		3.5308	1.1830	8	5	6.6874	

REGULAR ACTIVITY STATISTICS

ACTIVITY INDEX	AVERAGE UTILIZATION	STANDARD DEVIATION	MAXIMUM UTILIZATION	CURRENT UTILIZATION	ENTITY COUNT
1	0.0868	0.2816	1	0	66
2	0.4736	0.4993	1	0	61
3	0.2873	0.4525	1	1	24
4	0.4154	0.4928	1	1	27
5	0.6320	0.4823	1	1	81
6	0.0134	0.1149	1	0	61
7	0.4692	0.4991	1	0	969

MACHINE AND TRANSFER MECHANISM UTILIZATION

Figure 5.16. SLAM-Generated Summary Report

generates two basic statistics: memory (file) and processor utilization. The memory queues are identified by the computer module's event number in the file statistic report. Processor utilization is based on SLAM global variable XX (COMP**). These variables must be identified with the SLAM control TIMST statement [58]. Thus, each time-persistent statistic is identified by computer or process name.

Average delay statistics are automatically provided for the computer module's interrupt/pooling service routine. In Figure 5.16, file 10 shows the average delay time and the number of delayed I/O service requests.

I/O interface module statistics are identified by the ACTIVITY branch number in the regular activity statistics. AWAIT nodes and resource statistics are also provided for complex communication networks. All are identified by the appropriate resource or file number. The numerical identification scheme must be consistent and well structured. Process interface modules provide statistics according to SLAM branches and nodes used in the module configuration.

Observation statistics are generated by the SLAM COLCT subroutine. This subroutine can be employed in both the network and discrete portion of the model. For example, a COLCT node was placed at the cell's exit transfer device to determine average completion time for parts entering the cell. The time the part entered the cell is recorded in mark attribute ATRIB(6). The COLCT node records the elapsed time based on ATRIB(6). The statistics appear under the observation statistics in the SLAM summary report (Figure 5.16).

The manufacturing system statistics are generated in the same manner. Attributes and files related to the simulated processes should be well structured to avoid confusion with the computer control system. The simulated time unit must be considered carefully. The computer system which has a basic time unit of milliseconds is orders of magnitude different from the manufacturing process. Problems can occur in simulation variation statistics because of computer round-off error. Reference [58] presents a detailed description of SLAM-generated statistical collection and computation techniques.

Continuous Output

SLAM generates graphical output for continuous state variables. The RECORD and VARIABLE SLAM control are used to determine plot format output. Up to 10 state variables can be plotted with an independent variable (time). SLAM normally generates a line printer plot.

A preprocessor program was developed to read the SLAM-generated plot data file and transform it into the DEC Remote Graphics Instruction Set (ReGIS). The ReGIS data file is interpreted by a DEC General Imaging Generator and Interpreter (GIGI) terminal, which generates a SLAM continuous plot on a cathode-ray tube (CRT). Plots can be printed directly from the CRT, using the GIGI and a DECwriter IV graphics printer. (Other graphics terminals can also be used to generate SLAM continuous plots.) An example of a spring-mass-damper system output with step input disturbances and responses is shown in Figure 5.17.

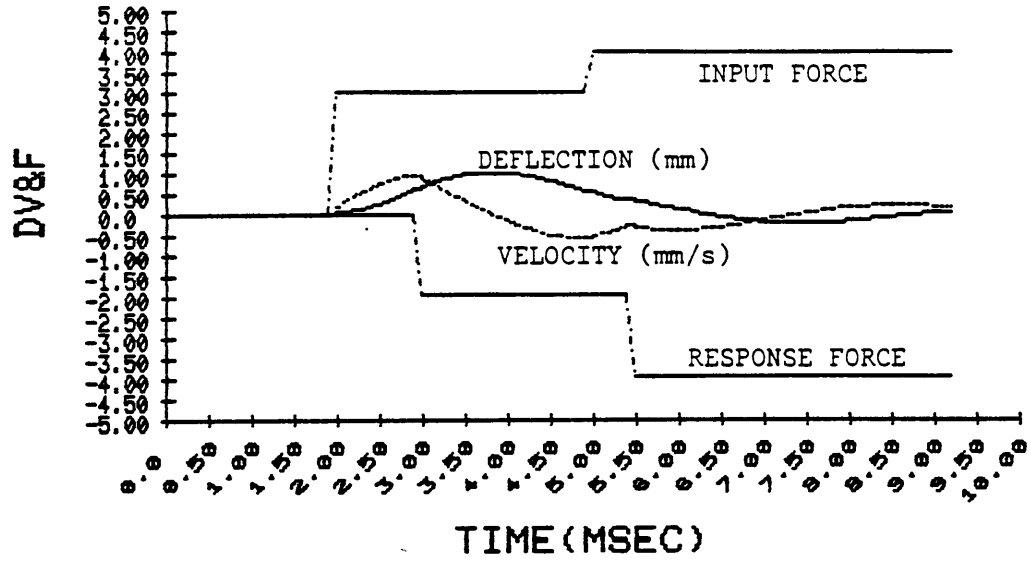

Figure 5.17. Spring-Mass-Damper System SLAM-Generated Continuous Plot

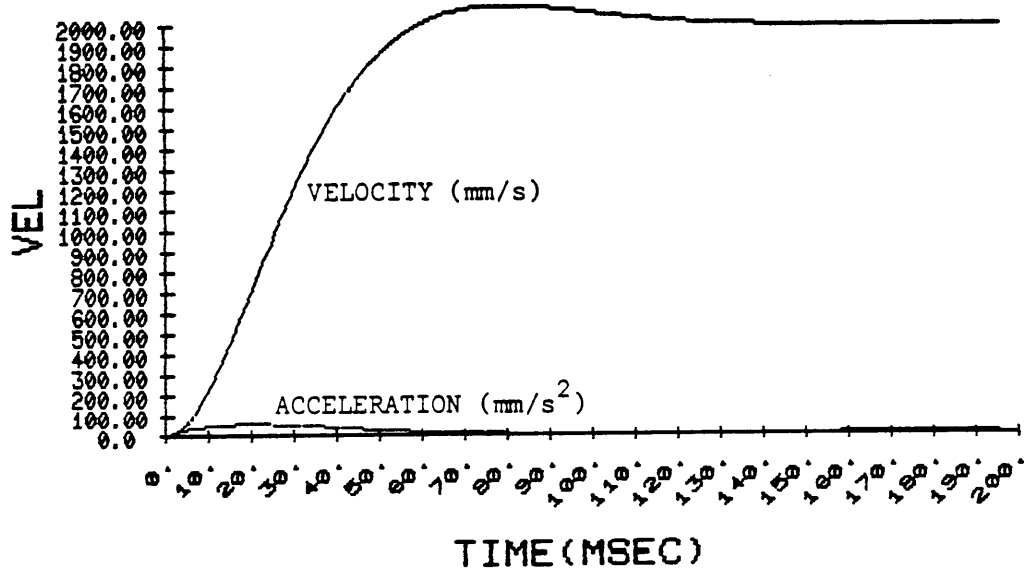

Figure 5.18. Continuous Plot of CNC Servo to Step Input

This continuous plot output capability enables analysis of complex control systems such as machine tool dynamics (Figure 5.18). Actual data collected from sensors on physical systems can be compared directly with the simulated state variable. Control actions and commands from the processor can also be displayed. Measured real-time digital performance data from the physical system can be "overlayed" directly with the simulated results, greatly increasing the analytical capability of the simulation model. Once the dynamic equations have been validated, the continuous process interface module can be "plugged into" the manufacturing system and overall system performance determined. Thus, the simulation model's structure captures the system variability and dynamics at the lowest possible level and enhances statistical analysis and validation.

System Development

An example manufacturing cell is developed and simulated with the three basic modules. The example cell demonstrates the simplicity and flexibility of the model structure, from its hierarchical control system to the cutting dynamics of a turning operation. Several simulation experiments are performed to illustrate the model's design and analytical capabilities.

The initial cell design is illustrated in Figure 5.19. Parts are transferred into the cell from the factory by conveyor. A storage queue is located at the end of the conveyor. Parts are removed from the cell through a separate output conveyor leading to other points in the factory. A stepping table serves the cell's input/output conveyors

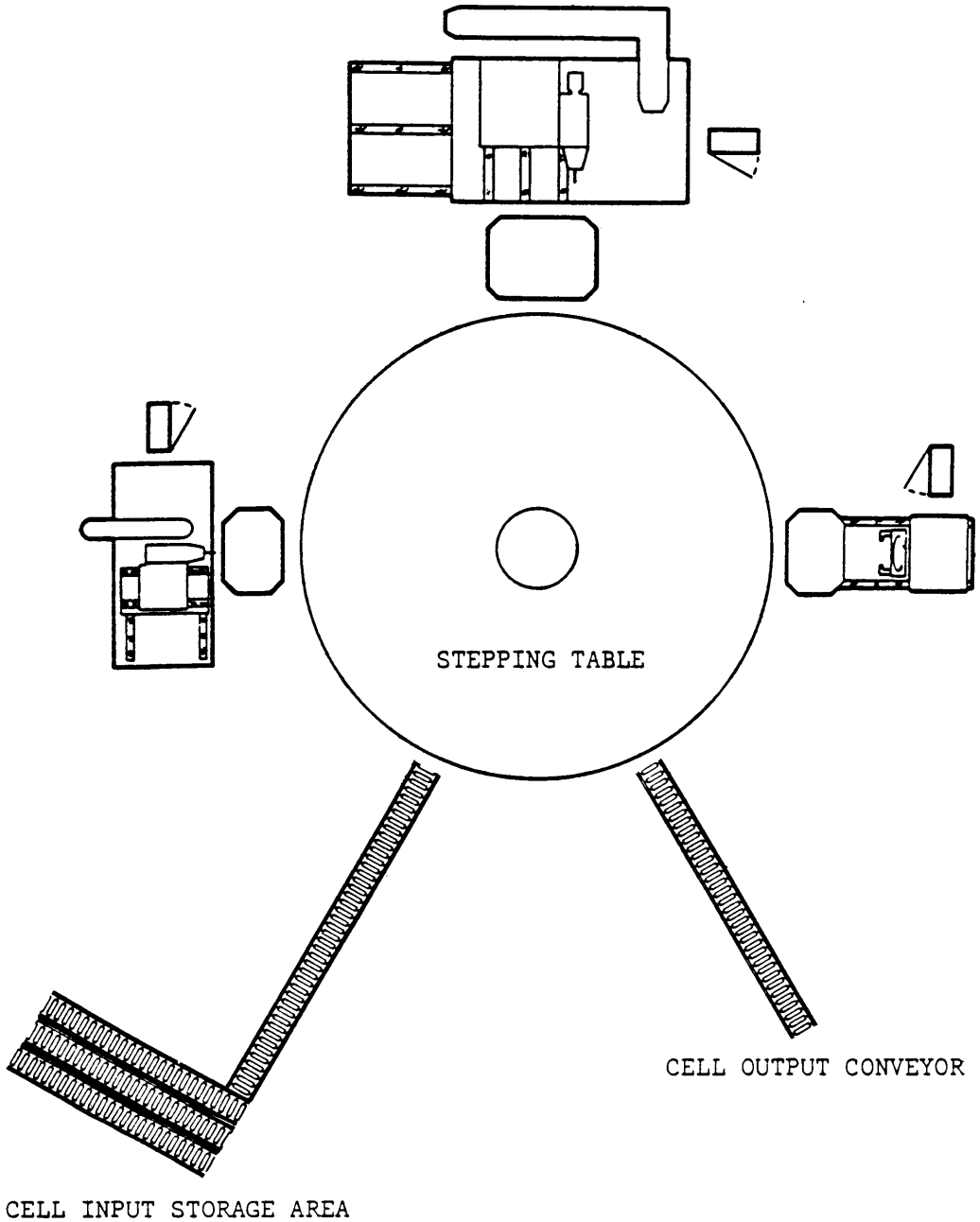

Figure 5.19. Example Manufacturing Cell Configuration

and machine stations, which have individual part transfer mechanisms. Initially, three machines are located in the cell to perform turning, milling, and drilling operations. Prefinished, cylindrical parts requiring turning, drilling, and milling operations are machined in this cell.

The cell control system and processes are simulated with the basic components of the model structure. Several sample analyses of the cell process and control system are performed to evaluate both the control system and manufacturing processes. The continuous portion of the model is used to evaluate an optimal adaptive control technique for a turning operation in Appendix C.

Process

The manufacturing cell shown in Figure 5.19 comprises a CNC lathe, CNC drill, and multipurpose CNC machining center. The cell is designed to process a family (or group) of parts requiring turning, drilling, and milling. Cell design and machine selection are a critical part of automated factory design. Cell configuration has a direct effect on the capacity and flexibility of the entire manufacturing system. Although the control system can enhance cell operation, basic process layout and machining capabilities determine production capacity. Thus, the physical configuration of the processes is the first and most critical step in automated system design.

A SLAM network representation of machine and transfer processes with interface modules is presented in Figure 5.20. The machine operations are represented by two ACTIVITY branches. The processes are identified by the following labels:

TR01 - input transfer device

MH02 - CNC lathe and transfer device

MH03 - CNC drill and transfer device

MH04 - CNC multipurpose machining center with transfer device

TR06 - output transfer mechanism

TR07 - stepping table

The actual physical processes are ACTIVITY branches. One ACTIVITY branch represents the transfer mechanisms; the other is the machining operations. Transfer completion times to and from the stepping table are normally distributed with means and standard deviations indicated on each transfer ACTIVITY branch.

Machining times are based on the part task and process. These times are computed in the discrete portion of SLAM and stored in ATRIB(4). The ACTIVITY branch then uses ATRIB(4) as the process time. By computing process times in the discrete portion of SLAM, many part/process time distributions can be sampled for the same machine. Setup and fixturing are also included in machine time.

Process and I/O Interface Modules

The process interface module is integrated into the actual process ACTIVITY branch. The GOON node receives the message event from the communication ACTIVITY branch. For the input, output, and stepping

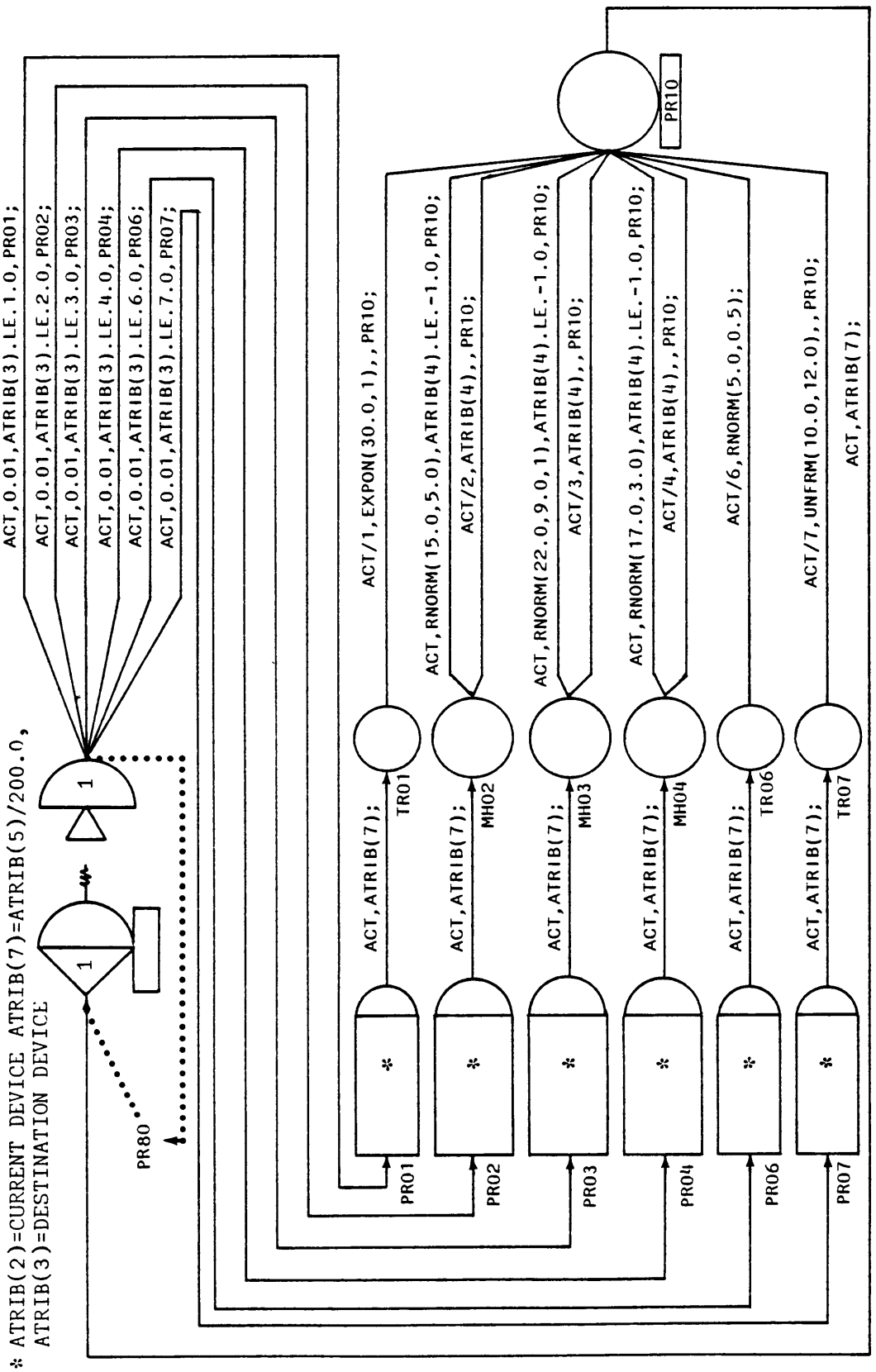

Figure 5.20. Cell Simulation Network

table activities, the message event passes directly to the ACTIVITY branch. Machine operations have two ACTIVITY branches: transfer and machine. A message event conditionally branches from the GOON node to the appropriate activity, based on ATRIB(4). If ATRIB(4) is negative, then the transfer activity occurs. Positive values of ATRIB(4) start machine activities. The value of ATRIB(4) is determined in the cell computer (COMP10) control logic.

Full-duplex communication is used in this example for clarity. The cell computer (COMP10) is connected to each process through an I/O interface module. Conditional ACTIVITY branches with communication delay times connect the cell computer to each process. The communication/processes are labeled with PR**, where ** is the process number. COMP10 sends message events through the ENTER node to the process (PR**) identified by ATRIB(3). The message event branches to the label specified by the ACTIVITY branch.

Each communication link has an ASSIGN node. The ASSIGN node sets the origin device attribute ATRIB(2) to the current device. The destination device attribute is set equal to the cell computer (COMP10). In addition, message process time is computed and stored in ATRIB(7). Thus, the control attributes are changed to send a message event back to COMP10 after the process ACTIVITY branch is completed.

The communication ACTIVITY branch directly preceding the ASSIGN node determines transfer time for the message event based on the message length, ATRIB(5). A 300-baud communication rate is simulated with four communication bits. At the end of the simulated

communication time, the process interface GOON node event occurs. Once the process is finished, a completion message event is sent to the cell computer GOON node. The return communication time is simulated using ATRIB(7). When the message transfer time activity is completed, the process EVENT node is encountered. Cell computer COMP10 determines the next operation command, based on the state of the manufacturing cell.

Communication ACTIVITY branches connect the cell computer with the next level computer (COMP80) in a similar manner. Message transfer times are determined by ASSIGN nodes. Control origin and destination attributes are determined in the computer's discrete software logic. Full-duplex communication is simulated between the two computers.

Computer Modules

The cell computer (COMP10) makes machine and transfer decisions for the entire cell. The cell computer also keeps track of all parts in the cell. A higher level factory computer (COMP80) monitoring all parts in the system sends message events to the cell computer (COMP10), immediately before the part arrives in the system. COMP10 makes a decision to accept or reject the part, based on the part operation requirements and cell status. COMP10 sends command message events to the CNC machines and transfer mechanisms based on the current cell state.

In the following sections, cell and manufacturing system logic and file configurations are presented. Cell control software is outlined, and message formats are presented. File configurations are developed for the cell computer. Finally, the higher level computer software and

random sampling procedure are defined. A complete program listing is provided in Appendix A.

Cell Software Logic

Cell computer (COMP10) logic is based on the device requesting service. A subroutine is written for each service request. When the computer is not servicing requests, it is considered idle.

Subroutine C10D80 services message events from the higher level computer (COMP80). The routine first checks whether the part tasks broadcast from COMP80 can be machined by the cell. If the part is accepted by the cell computer, a message event is sent to COMP80 to release the part to the cell's input conveyor. The part and tasks are stored in file 1 according to rank attribute. In the initial cell, parts are ranked and processed FIFO.

The computer checks the status of the input transfer mechanisms, stepping table, and input table position. If the input position on the table is not occupied and the table transfer mechanism is idle, the next part in COMP10's file 1 (memory queue of parts) is transferred onto the stepping table. COMP10 sends a message event to device TR01 to initiate the transfer. Otherwise, a transfer does not occur.

Once the cell input transfer has occurred, a message event is received by COMP10 from device TR01. Subroutine C10D01 services the message event. A call is made to MEMSTO to determine which machines can perform the current task, as identified by the task pointer. Machines and estimated machining times are identified for the task from the data array in the computer memory. The part number is stored in

the cell computer memory file(s) having the same identification number(s) as the machine(s) able to perform the part's next task.

Subroutine SERCHK is called to check the status of all other transfer devices. If all transfers have been completed, a message event is sent to the stepping table interface module (PR07). The stepping table transfers the parts in the cell. At the end of a step, the stepping table (TR07) sends a message event to COMP10. Subroutine C10D07 is called. The subroutine updates the position of all parts on the stepping table. At the same time, a check is made of each machine memory file in the cell computer. If the next part indicated in the file is the same as the part on the table, transfer logic is executed by a call to subroutine TABSER. TABSER checks the status of the machine. If the machine is idle, a transfer message event is initiated for the appropriate machine, using the negative value of the estimated machine time in ATRIB(4) to indicate a transfer. A transfer cannot occur if the part on the machine is busy or the part on the table is not in the machine's cell computer file. The same procedure is used to initiate transfers into and from the cell. All devices having access to the stepping table are checked by TABSER. Control is returned to subroutine C10D07, where a call is made to subroutine SERCHK to check for the next table transfer.

A machine transfer completion event is serviced by C10D02. Since all the machines in the cell are CNC, this subroutine services every machine transfer and machine completion message event. If ATRIB(4) is

negative, the transfer completion is identified by the subroutine. A simulated processing time is sampled from an appropriate distribution and stored in ATRIB (4). A message event is sent to the machine requesting service with ATRIB(4) containing a machine activity time. Once the message event is sent to begin machining operations, the part identification number is removed from other computer machine files having the same part and task. A call is made to SERCHK to check cell status for advancing the stepping table.

A machine completion event is also serviced by C10D02. A positive ATRIB(4) indicates machine completion, and machine status is changed to idle. The part pointer is incremented and the next task identified. Three states are considered with the following actions.

1. The current machine can perform the next task, the part is kept on the machine, and a message with the machine time is sent to the machine.
2. Other machines in the cell can perform the next task, a call to MEMSTO is made to store the part identification number in the machine's cell computer memory, and a transfer command message is sent to the current machine station.
3. Cell machines cannot perform the next task, the part identification number is stored in the output memory file to schedule the part for the cell output device, and a transfer command is sent to the machine.

In all cases, stepping table status is checked. A call to subroutine TABSER is made for a possible part transfer, based on the

table part number, as described previously. The logic continues until all parts to the cell have been served.

An example message format is presented in Figure 5.21. This message event contains the part number ATRIB(8) and task pointer ATRIB(10). Parts are processed by manufacturing part priority. In the example, the pointer indicates that task 8 is next to be performed. If task 8 cannot be performed by the example cell, the part is sent back into the manufacturing system. This task-oriented system can identify many machines for the same part operations, adding flexibility and reliability to the manufacturing cell.

Cell Computer Files

CNC machines, transfer devices, microprocessors, and minicomputers control all process operations. The cell computer determines the sequence and timing of task operations. Machine and transfer device microcomputers control the actual physical movements by accessing task programs. Thus, cell computer control software development time and memory requirements are greatly reduced. This reduction in software overhead was identified by the mathematical model in Chapter 3. The mathematical model optimal solutions identified all CNC machines at the bottom of the hierarchy. If DNC machines were used, a more complex control software structure would have to be developed, providing complicated interrupt service routines for each machine and transfer device.

A file is used for each machine and transfer device in the cell's memory. File allocations are shown in Figure 5.22. A simple file

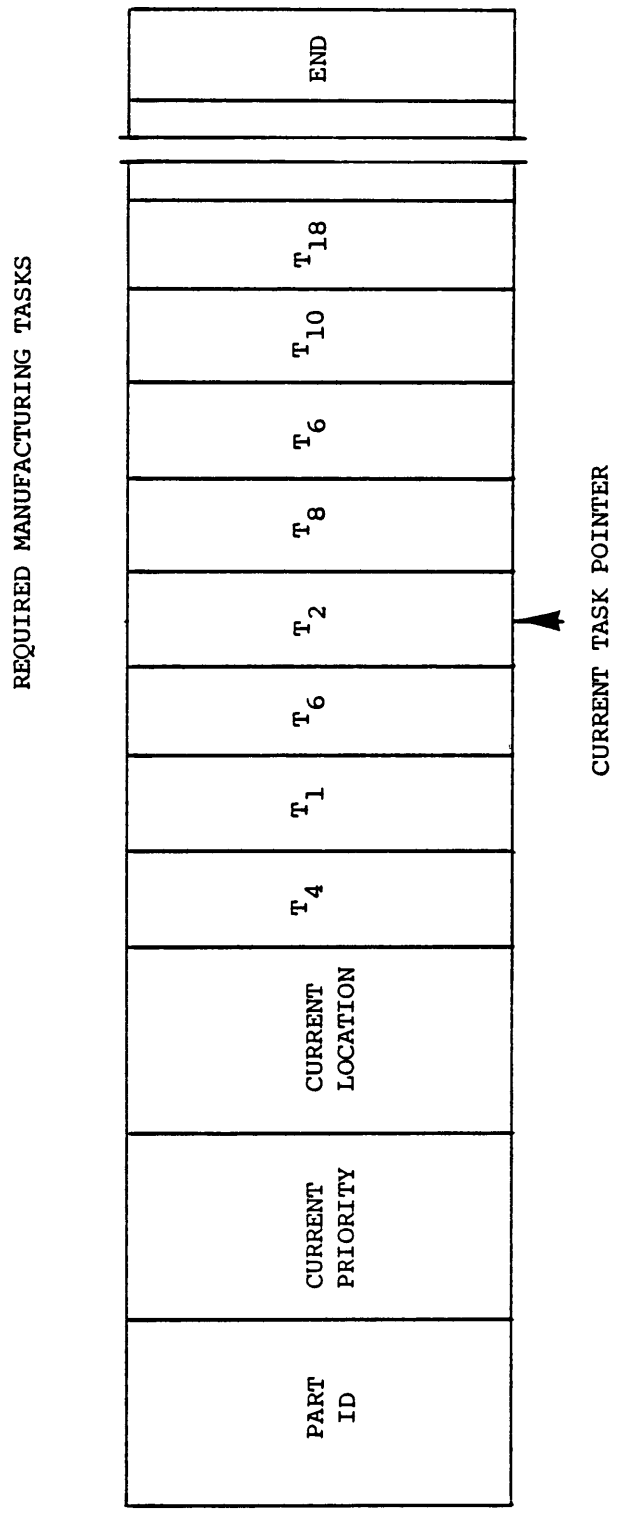

Figure 5.21. Task Message Format

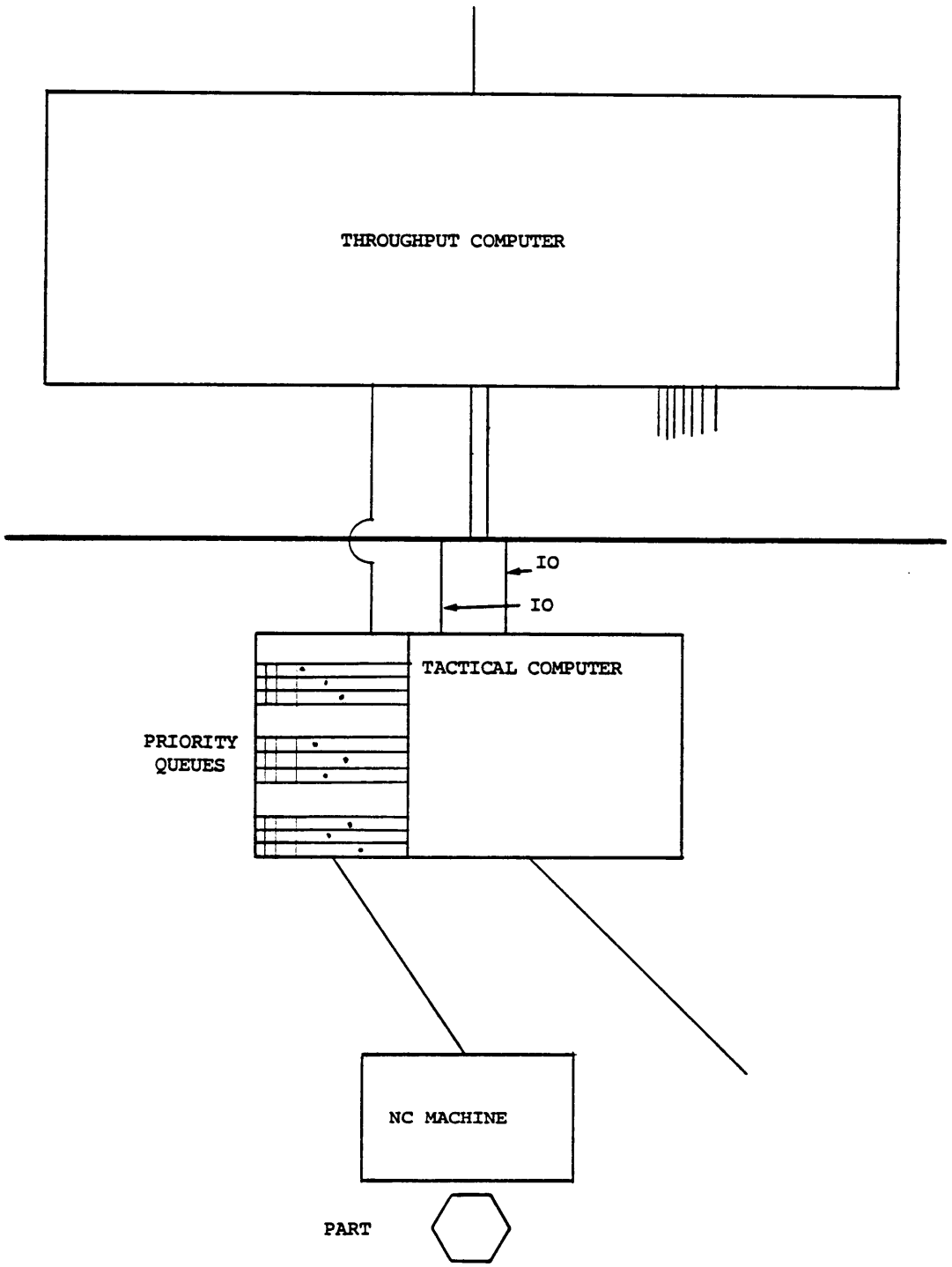

Figure 5.22. Cell Computer Module Memory Configuration

structure is possible, because all devices are "smart" and do not require simulated computer modules.

Files are ranked on a single attribute, FIFO or LIFO. SLAM control statements determine the ranking of each file as indicated in Figure 5.22. The entire ATRIB array for each part is stored in a file. These files represent the part's dynamic ordering in the cell computer's memory. Thus, parts can also be dynamically scheduled for several machines according to part manufacturing priority, ATRIB (9). As the table is stepped, transfer decisions are based on the first part identification number in the memory file. If the table has the part identified in memory, the part is transferred; the part identification number is then removed from memory files of other machines which can perform the same task.

This file system takes advantage of the SLAM file statement subroutines. All parts can be located in the files with several functions. File status, such as number of entries, can be checked with simple SLAM functions. In addition, the files are ranked automatically as entries are put into files. This ranking is determined by a single control statement. When an entry is removed from the file, all attributes are stored in the ATRIB array. The array values are the same as the message format. Statistics are also recorded for each file.

Higher Level Computer Software

The higher level computer sends message events to all cell computers on a "broadcast" type system. In this example, the higher

level computer generates parts and task message events for the cell computer (COMP10). COMP10 determines whether the part task can be performed by any of the machines in the cell.

Initially, cell part types are sampled from a uniform distribution. Tasks associated with the part type are stored in the message event ATRIB array. The task pointer is set to the first task. Each part is given an identification number. A message event is sent to the cell computer. The next part event is scheduled, based on an exponential arrival time with a mean of 210 seconds. The process is continued until the end of the simulation experiment.

The main emphasis and evaluation are based on the cell and machine operations. The higher level computer (COMP80) is a part generation cell to test cell computer (COMP10) software logic. In an actual manufacturing system, the higher level computer would be programmed to control cell computers. Discrete SLAM control programs and system software can be developed in a similar manner as the cell computer module.

System Analysis

Analysis of the results of simulation experiments is a nontrivial task. The simulation model represents a stochastic process, and random elements of the model produce probabilistic output. As in a real system, the variability inherent in the simulation model must be determined. The advantages of the simulation model are twofold: parameters can be estimated for distributions of random variables, and performance variables are readily obtained from experimental results.

After a simulation model has been developed, experiments have to be designed to provide a basis for evaluating system performance variability between and among different system configurations and operations. Performance variables must be chosen to make these inferences. System performance characteristics are random variables and are selected on the basis of system configurations and/or operations being investigated. System performance inferences using sample statistics are made extremely difficult because of the inherent covariance in a computer-controlled system. Much of the covariance comes from a prior knowledge of part routing and control actions. Thus, the actual computer control system is not a stationary stochastic process, as assumed in the simulation model.

An analysis is performed to illustrate the model's capability in measuring system and component performance. The analytical method presented is consistent with the overall evaluation methodology. This approach is not absolute, and other analytical techniques can be employed [16]. However, the analytical procedure is very useful for selecting hardware components and evaluating system software control procedures.

Analysis Matrix

Many types of configurations and operational procedures can be analyzed with the model structure. Statistics can be generated for almost every type of performance (random) variable under consideration. The major problem is setting up the experimental designs to analyze system performance, using estimates of these random variables. A basic

approach for analyzing system performance is developed to enhance the experimental design.

The manufacturing system can be separated into two basic areas: physical components and software control. A component/software analysis matrix (Figure 5.23) is developed from these two major categories. The columns represent physical control and equipment options to be evaluated based on system performance. The rows are software control options. Performance random variables are selected for the statistical inference performance tests related to the various component/software combinations identified in the matrix.

A sequential search technique is used to find an optimal component/software combination. The search begins at the least expensive control procedure and the initial system components identified by the mathematical model. Using statistical inference tests, the initial system configuration is compared with the adjacent hardware or software control. If an improvement is detected at a given level of confidence, the new control logic or configuration is considered the base design and compared with other configurations in a similar manner. By arranging the different system configurations and control software in order of increasing cost, the search technique can be used to make economic/performance decisions at each evaluation point in the search procedure. Some alternatives under consideration can have both software and hardware improvements. In these situations, the alternative should be placed according to the highest cost software and hardware component.

SOFTWARE	RANDOM VARIABLE	PHYSICAL HARDWARE				
		Basic System	Add Machine	Adaptive Control	Robot Transfer	. . .
FIFO	Time in cell					
	Machine utilization					
	. . .					
NAM	Time in cell					
	Machine utilization					
	. . .					
SNOP	Time in cell					
	Machine utilization					
	. . .					
SEPT	Time in cell					
	Machine utilization					
	. . .					

FIFO (first-in first-out) - all machines
NAM (next available machine)
SNOP (shortest number of operations)
SEPT (shortest estimated processing time)

Figure 5.23. Component/Software Analysis Matrix

For the example cell, six component and software control combinations are investigated, as shown in Figure 5.23. Two performance variables are considered: machine utilization and part time in the cell. Since both means and variances are estimated for each step of the search technique, the T-statistic is used for the hypothesis testing between the two means for both random variables. The null hypothesis is that the mean of the base component/software combination is less than or equal to the mean of the next combination under consideration in the search method:

$$H_o: \mu_{BASE} \leq \mu_{SEARCH}$$

$$H_1: \mu_{BASE} > \mu_{SEARCH}$$

If H_o is rejected for the part time in cell random variable, then the search component/software combination is considered the new base. Otherwise, the base combination is used for the next step in the search. More detailed analysis of variance tests must be performed for conflicting random variable hypothesis test results [70].

Since the cell requires less than 30 seconds of CPU time per eight hours of simulation time, an eight-hour simulation run is considered a replicate for the hypothesis testing. The number of replicates required is based on the confidence interval and power of test required. Initially, twenty runs are performed.

The number of additional runs is based on the pooled variance, level of significance for the statistical test α, and the difference between the two means detected with a power $1-\beta$. The sample size required is computed from:

$$z_{\beta} = \frac{-\lambda + t_{1-\alpha/2}}{\sqrt{1 + (t^2_{1-\alpha/2})/(4n-4)}}$$

where:

$\lambda = \sqrt{n}d/s_p$

s_p - pooled estimate of the variance

$d(>0) = (\mu_{BASE} - \mu_{SEARCH})$

$t_{1-\alpha/2}$ has (2n-2) degrees of freedom

n - number of replications

Once the required number of replications is run, the hypothesis tests are performed and the search continues. When no significant performance improvements can be found, the search ends. The performance random variables are assumed to be independent and identically distributed. (This assumption might not be valid for all simulation experiments.)

Example Cell Operations

The basic cell for the sample analysis can perform the tasks listed in Table 5.1. The initial cell has three machines: (1) CNC lathe, (2) CNC drill, and (3) CNC machining center. As mentioned in the previous section, final turning, drilling, and milling operations are performed to produce variable-diameter shafts with different drilling and end milling requirements. The stepping table and transfer devices can handle the different tasks required of this part group. Machine times are assumed to be exponentially distributed.

Table 5.1

Task Sequence and Estimated Task Processing Times

PART TYPE	TASK SEQUENCE
1	18, 22, 20
2	8, 21, 12
3	18, 22, 10
4	8, 21, 4, 12
5	12, 21
6	12
7	18, 20
8	8, 18, 22

POSSIBLE TASK ASSIGNMENT

TASK NUMBER	MACHINE NUMBER	ESTIMATED PROCESSING TIME (in seconds)
4	4	120
	5	140
8	2	095
	5	105
10	4	300
12	1	250
18	1	180
20	4	390
	5	275
21	2	300
22	2	345

Part requirements are generated from the higher level computer using a uniform distribution. When any of the eight different parts in the group is selected out of 25 total parts, the cell computer accepts the part. The task sequence for each part type in the group is shown in Table 5.1. Thus, uniform loading of all part types is initially assumed. Time between part requests is exponentially distributed with a mean of 210 seconds. The basic simulation time unit is 1 second.

Process times of computers and processors are assumed constant without random variation. Cell computer process times are three to six orders of magnitude smaller than the machine process times, making any variation insignificant. The software logic is verified using the state table output, as shown in Figure 5.16.

Experiments

The software control and one hardware configuration are investigated using the example previously described. Each control strategy is described as follows.

1. FIFO: All parts are transferred and machined FIFO, based on their arrival to the cell and machines. Machine processing order is based on the time the part is scheduled for the machine.
2. NAM (next available machine): A part on the stepping table is transferred to any idle machine which can perform the part's next task, regardless of the memory queue ordering. Parts entering the cell are served on a FIFO basis.
3. SNOP (shortest number of operations): Parts are serviced by the cell based on shortest number of operations.

Finally, an extra lathe is added to the cell. This lathe has the same machining characteristics as machine 2.

The experiment begins with FIFO control strategy and one lathe. This procedure causes a low machine utilization because parts on the transfer table are "blocked" from empty machines due to the file processing order. Only one part was completed during each eight-hour simulation run. The search was immediately directed to the next software alternative.

Next, the NAM procedure is implemented in the cell (Table 5.2). This control strategy was selected over addition of a new machine, since changing four lines of control system source code was less expensive than the cost of a new machine. Results show significant improvement over the FIFO method, as the average time a part spent in the cell was 1720 seconds.

SNOP was implemented next with three machines. The time-in-cell hypothesis test could not be rejected. NAM machine utilization was also significantly greater than the SNOP option. NAM remained the base software/component configuration. The next step in the search was to evaluate the physical/hardware component.

In the NAM control strategy, machine 2 memory queue length is estimated to be twice as high as for any other machine. This indicates that another machine is required. Thus an extra machine was added to the SNOP option, reducing average part production time significantly. However, machine utilization decreased with a much lower standard deviation. The SNOP option with an extra machine 2 becomes the base

Table 5.2

Example Search Analysis

STEP 1 - DIRECTION: SOFTWARE CONTROL

μ_{BASE}:FIFO μ_{SEARCH}:NAM $n = 20$, $\beta = 0.2$, $\alpha = 0.1$

FIFO has unacceptable throughput rate of 1 part per 8 hours
Accept: NAM

STEP 2 - DIRECTION: SOFTWARE CONTROL

	μ_{BASE}:NAM		μ_{SEARCH}:SNOP		
	$\bar{X}$	s	$\bar{X}$	s	d
Time in Cell(s)	1721	1646	1281	1182	300

$n = 20$ $f = 38$ $t_{0.90}(38) = 1.69 \leq 0.97 = t_{exp}$

Cannot Reject: Keep NAM

	$\bar{X}$	s	$\bar{X}$	s	d
Average Cell Machine Utilization	0.53	0.151	0.42	0.148	0.05

$n = 20$ $f = 38$ $t_{0.90}(38) = 1.69 < 22.1 = t_{exp}$

Reject: NAM has greater machine utilization

STEP 3 - DIRECTION: HARDWARE

	μ_{BASE}:NAM (3 machines)		μ_{SEARCH}:NAM (4 machines)		
	$\bar{X}$	s	$\bar{X}$	s	d
Time in Cell(s)	1721	1646	1012	830	300

$n = 20$ $f = 38$ $t_{0.90}(38) = 1.69 \leq 4.18 = t_{exp}$

Reject: NAM and 4 machines

	$\bar{X}$	s	$\bar{X}$	s	d
Average Cell Machine Utilization	0.53	0.151	0.44	0.02	0.05

$n = 20$ $f = 38$ $t_{0.90}(38) = 1.69 < 77.5 = t_{exp}$

Reject: NAM and 3 machines have greater machine utilization

configuration because of the significantly shorter time in the cell. Another step was made in the software control direction without any detectable improvement in the performance variables.

The simulation model could be extended to investigate the increased cost of the machine versus the savings from increased cell production rate and decreased variability in part process time. The testing can continue from this point. Stopping rules, such as diminishing performance increases or increasing costs, can be used to end this type of analytical procedure.

This example illustrates the importance of capacity and performance limitations determined by the manufacturing process. Control strategies can improve system performance only up to these limits.

Summary

A comprehensive simulation model is developed for evaluating design and operational aspects of a manufacturing system. The model incorporates a modular structure to build a control network. SLAM's continuous modeling capability enables machining system dynamics to be investigated, as shown in Appendix B. Variations from the process level can be evaluated in conjunction with the entire production system and control network.

The model can evaluate software logic and emulate control actions for the entire factory in real simulation time. Computers are simulated with interrupt capability. Communication systems can be evaluated. All components and software logic are developed similarly

to the actual physical system. Finally, output from the model provides all the statistics and information necessary to fine tune the basic systems identified by the mathematical models. This fine tuning involves determining the operating procedure and component dynamics of the actual physical design. This modeling structure can simulate and analyze most automated manufacturing systems at virtually every level and degree of detail.

CHAPTER SIX

CONCLUSIONS

A comprehensive methodology is developed for the design and evaluation of all process levels in a manufacturing system. The entire methodology is based on the manufacturing process being controlled. All control systems are determined by manufacturing process requirements. The methodology not only identifies optimal hardware configurations, but also evaluates hardware and software performance characteristics of these configurations with respect to the dynamics of the manufacturing processes.

The methodology is enhanced by using both mathematical and simulation modeling. The mathematical model identifies an initial optimal configuration for the control system and can be applied to all levels of the manufacturing system. This optimal solution is based on process control requirements, control component costs, and performance characteristics. Solutions to the mathematical model are readily obtained by using dynamic programming and finite component cost/performance relationships. In addition, the model helps to delineate system boundaries and determine initial control and performance variables.

The simulation model analyzes the dynamic and stochastic nature of the manufacturing system identified by the mathematical model. The simulation language structure enables hardware, software, and control processes to be simulated simultaneously. This is the first simulation

model structure in which control logic execution and process responses can be evaluated in a real-time context. All computer I/O techniques, including interrupts, can be simulated by the model. Modularity of the structure enables different components to be added to the simulation network. The model can simulate analog processes and produce graphic output of the simulation experiments. Continuous simulation output can be validated against output from the actual physical process (e.g., strip charts). Finally, the statistical and trace model results enable both model validation and performance analysis.

Results from the simulation model can be used for design or mathematical model validation. Through postoptimality analysis, the mathematical model can identify updated optimal configurations based on simulation results, just as the simulation model can be used to evaluate results from actual system operation.

This methodology can be employed at all phases of the manufacturing process. The tool designer can evaluate tool performance characteristics. Manufacturing engineers can evaluate the production system. Computer software and communication hardware can be analyzed with respect to system performance. All control system designs are based entirely on manufacturing requirements and performance. The methodology can significantly reduce manufacturing system planning and design time.

Initial Results

Several findings concerning the design, control, and operation of an automated manufacturing system were made during the development and initial application of the methodology. These results confirm the following control concepts.

Design of Manufacturing System

The design and layout of the physical process determines the capacity and optimal operating limits of the manufacturing system. Computer control can improve system performance only within these operating limits. Additional machine capacity, resulting in lower machine utilization, is required in flexible manufacturing systems to reduce system variability caused by the wide range of part operations and lot sizes.

Distributed Control

Control decisions should be made as close to the process as possible. This reduces software costs and increases control system capacity. Communication message and processor execution rate requirements are greatly reduced, further decreasing hardware costs. Distributed control also increases system reliability.

Dynamic Scheduling

Process decisions should be made as close to their execution time as possible. This enables process control actions to be executed based on the current system state. Dynamic scheduling is extremely important in flexible manufacturing systems because the part mix, priorities, and

machining conditions can change rapidly, requiring instantaneous decisions based on current and projected system states. Properly designed computer control systems can provide this capability.

Compensatory Tracking Optimal Control

The compensatory adaptive control system has great potential for optimal dynamic control of CNC processes. The speed of control decisions obtained from direct register manipulation by the tracking algorithm procedure offers great potential for robotic systems.

This optimal control method can reduce process cycle time, while improving system performance. In addition, control computer I/O overhead is drastically reduced.

Future Research

Results identified in the preceding section are initial. Further investigation and analysis are required. The following research activities would extend the methodology's capabilities and improve manufacturing system evaluation:

1. validation of the mathematical modeling process with several manufacturing system applications,
2. extension of the methodology to robotics for component design and control,
3. development of a simulator with network interactive graphics for manufacturing system design and evaluation,
4. design of an input measurement system to validate continuous simulation results with actual physical data, and

5. implementation of the simulator on a microprocessor or minicomputer to facilitate its use as a comprehensive design tool.

The flexibility and capability of the methodology allow it to be applied to many areas of research in computer-controlled manufacturing systems. The methodology makes it possible to integrate manufacturing process control and computer system design.

REFERENCES CITED

1. Aishton, Thomas H., and Miller, Richard A. "Evaluation of Physical Modeling as a Tool for the Design and Implementation of Computerized Manufacturing Systems." Proceedings of the 1982 Annual Industrial Engineering Conference , New Orleans, Louisiana, May 23-27, 1982. Norcross, Georgia: Industrial Engineering and Management Press, Institute of Industrial Engineers, 1982. Pp. 411-418.

2. Allen, Bruce S. "Data Highway Links Control Equipment of Any Number of Different Manufacturers." Control Engineering 28, no. 7 (July 1981): 87-90.

3. Analog Devices. Analog-Digital Conversion Handbook. Norwood, Massachusetts: Analog Devices, 1981.

4. Bailey, S. J. "Precise Positioning: A Matter of Sensors and Loop Dynamics." Control Engineering 27, no. 12 (December 1980): 50-54.

5. Bibbero, Robert J. "Simulating a Control System's Data Highway Involves a Careful Choice of Computer Models." Control Engineering 28, no. 7 (July 1981): 81-84.

6. Blumberg, D. F. "A Production Analysis Simulation System (PASS)." Proceedings of the Winter Simulation Conference. 1978.

7. Bollinger, John G., and Crookall, John R. "The Role of Simulation in Design/Teaching of Manufacturing Systems." Annals of the CIRP 30 (1981): 525-532.

8. Brayer, Kenneth; LaFleur, Valerie; and Simpson, Gary. "Simulation VIA Implementation with Applications in Computer Communication." Fifteenth Annual Simulation Symposium, IEEE Computer Society, Tampa, Florida, December 1979. Silver Springs, Maryland: IEEE Press, 1980. Pp. 239-264.

9. Bronner, L. "Overview of the Capacity Planning Process for Production Data Processing." IBM Systems Journal 19 (1980): 2-27.

10. Buzacott, J. A., and Shanthikumar, J. G. "Models for Understanding Flexible Manufacturing Systems." AIIE Transactions (December 1980).

11. Childs, James J. Principles of Numerical Control. New York: Industrial Press, 1970.

REFERENCES CITED (CONTINUED)

12. Colwell, L. V.; Frederick, J. R.; and Quackenbush, L. J. Research in Support of Numerical and Adaptive Control in Manufacturing. Ann Arbor, Michigan: University of Michigan, 1969.

13. Davis, Robert P.; Wysk, Richard A.; Agee, Marvin H.; and Kimbler, Delbert L. "Optimizing Machine Parameters in a Framework for Adaptive Computer Control." Computers and Industrial Engineering 4, no. 2 (1980): 143-151.

14. Fleming, Robert H., and Berry, Margaret. "Hierarchical Multi-Granular Modeling to Improve Software Performance." Fifteenth Annual Simulation Symposium, IEEE Computer Society, Tampa, Florida, December 1979. Silver Springs, Maryland: IEEE Press, 1980. Pp. 223-238.

15. Frost-Smith, E. H. "Optimization of the Machining Process and Overall System Concepts." Annals of the CIRP 19 (1971): 385-394.

16. Gibra, Isaac N. Probability and Statistical Inference for Scientists and Engineers. Englewood Cliffs, New Jersey: Prentice-Hall, 1973.

17. Gomaa, Hassan. "A Simulation-Based Model of a Vertical Storage System." Record of Proceedings, Twelfth Annual Simulation Symposium, Tampa, Florida. 1976. Pp. 273-301.

18. Groover, Mikell P. Adaptive Control and Adaptive Control Machining. West Lafayette, Indiana: Purdue University, 1977.

19. Groover, Mikell P. Automation, Production Systems, and Computer-Aided Manufacturing. Englewood Cliffs, New Jersey: Prentice-Hall, 1980.

20. Gupta, Parveen. "Multiprocessing Improves Robotic Accuracy and Control." Computer Design 21, no. 11 (1982): 169-176.

21. Haigh, Peter L. "A Methodology for Simulating Computer Systems." IBM Systems Journal 18 (1979): 277-314.

22. Ham, I. "Introduction to Group Technology." Technical Report MMR76-03. Dearborn, Michigan: Society of Manufacturing Engineers, 1976.

23. Harrison, Howard L., and Bollinger, John G. Introduction to Automatic Controls. New York: Harper & Row, 1969.

REFERENCES CITED (CONTINUED)

24. Hartwig, Glenn C. "Electronic Control Moves into the Ascendancy: 1960-1969." Manufacturing Engineering 88 (1982): 181-202.

25. Heck, Robert. "Simulation: A Tool for Evaluating Computer System Software." Fifteenth Annual Simulation Symposium, IEEE Computer Society, Tampa, Florida, December 1979. Silver Springs, Maryland: IEEE Press, 1980. Pp. 91-98.

26. Hitomi, Katsundo. "Analysis of Production Models Part I: The Optimal Decision of Production Speeds." AIIE Transactions 8 (March 1976): 96-105.

27. Holmes, Lowell L. "Adaptive Control for Metalworking Applications." Automation 14 (August 1967): 77-82.

28. Hutchinson, George K., and Wynne, Bayard E. "A Flexible Manufacturing System." Industrial Engineering 12 (December 1973).

29. Intel. The 8086 Family User's Manual. Santa Clara, California: Intel, 1979.

30. Intel. MCS-80/85 Family User's Manual. Santa Clara, California: Intel, 1979.

31. Iwata, K.; Murotsu, Y.; and Oba, F. "Optimization of Cutting Conditions for Multi-Pass Operations Considering Probabilistic Nature in Machining Process." Journal of Engineering for Industry 95 (1976): 1-8.

32. King, Kenneth L. "A Building Block Approach to Advanced Control Techniques." Control Engineering 28, no. 9 (August 1981): 17-21.

33. Kompass, Edward J. "A Long Perspective on Integrated Process Control Systems." Control Engineering 28, no. 9 (August 1981): 4-9.

34. Kompass, Edward J. "The Pressure's On for Good Systems Design." Control Engineering 27, no. 10 (October 1980): 63.

35. Koren, Y. "Design of Computer Control for Manufacturing Systems." Journal of Engineering for Industry 101 (August 1979): 326-332.

36. Koren, Y. "Flank Wear Model of Cutting Tools Using Control Theory." Journal of Engineering for Industry 100 (February 1978): 103-109.

REFERENCES CITED (CONTINUED)

37. Ledgerwood, Byron K. "Trends in Control." Control Engineering 27, no. 9 (September 1980), 236.

38. Leventhal, Lance A. Introduction to Microprocessors: Software, Hardware, Programming. Englewood Cliffs, New Jersey: Prentice-Hall, 1978.

39. Lewis, C. D., and Foo, A. L. "GIPSI: A General Purpose Inventory Policy Simulation Package." International Journal of Production Research 18 (1980): 73-82.

40. Lewis, W. C.; Barash, M. M.; and Solberg, J. J. "Data Flow CMS Control System Architecture." Report No. 18, National Science Foundation Grant No. APR74 15256. West Lafayette, Indiana: Purdue University, December 1980.

41. "LSI Manufacturing Line Uses 100 Automated Tool Subsystems." Control Engineering 27, no. 5 (May 1980): 47.

42. Lukas, Michael P., and Willey, Michael S. "An Advanced System Architecture for Distributed Control." Control Engineering 28, no. 9 (August 1981): 10-15.

43. Maani, K. E., and Hogg, G. L. "A Stochastic Network Simulation Model for Production Line Systems." International Journal of Production Research 18 (1980): 723-739.

44. Masory, Oren, and Koren, Yoram. "Adaptive Control System for Turning." Annals of the CIRP 29 (1980): 281-284.

45. Medeiros, D. J.; Sadowski, R P.; Starks, D. W.; and Smith, B. S. "A Modular Approach to Simulation of Robotic Systems." Proceedings of the 1980 Winter Simulation Conference, Orlando Florida, December 3-5, 1980.

46. Merchant, M. E. "The Inexorable Push for Automated Production," Production Engineering (January 1977): 45-46.

47. Mitchell and Gauthier Associates. ACSL: Advanced Continuous Simulation Language User Guide/Reference Manual. Concord, Massachusetts: Mitchell and Gauthier Associates, 1975.

48. Mize, J. H. "Prosim V: A Production Systems Simulator." Proceedings of the Third Conference of Application of Simulation. 1969.

REFERENCES CITED (CONTINUED)

49. Morgan, Christopher L., and Waite, Mitchell. 8086/8088 16-Bit Microprocessor Primer. Peterborough, New Hampshire: BYTE/McGraw-Hill, 1982.

50. Morris, H. M. "A Microprocessor in Every Control Loop." Control Engineering 27, no. 9 (September 1980): 99-104.

51. Nguyen, H. C.; Ockene, A; and Skwish, W. J. "The Role of Detailed Simulation in Capacity Planning." IBM Systems Journal 19 (1980): 81-101.

52. Nutt, G. J. "A Case Study of Simulation as a Computer System Design Tool." Computer 11 (1978): 31-36.

53. "Performance Measurement Techniques for Adaptive Control Process Control." Final Technical Report, AFML-TR-68-265, MMP Project 8-323. Wright-Patterson Air Force Base, Ohio: Air Force Materials Laboratory, September 1968.

54. Peterson, James I. Computer Organization and Assembly Language Programming. New York: Academic Press, 1979.

55. Phillips, D. T.; Hardwerker, M.; and Hogg, G. L. "GEMS: A Generalized Manufacturing Simulator." Computers and Industrial Engineering 3 (1979): 225-232.

56. Poppendieck, Mary B. "Is Computer Control Worth It? Payoff Depends on Planning." Control Engineering 27, no. 12 (December 1980): 58-60.

57. Pressman, Roger S., and Williams, John E. Numerical Control and Computer-Aided Manufacturing. New York: John Wiley & Sons, 1977.

58. Pritsker, A. Alan B., and Pegden, Claude Dennis. Introduction to Simulation and SLAM. West Lafayette, Indiana: Systems Publishing Corporation, 1979.

59. Pritsker, A. Alan B., and Sigal, C. Elliott. Management Decision Making: A Network Simulation Approach. Englewood Cliffs, New Jersey: Prentice-Hall, 1983.

60. "Programmable Precision Manufacturing System (PPMS) for Small Lot Production." Technocrat 15, no. 2 (February 1982): 45-49.

REFERENCES CITED (CONTINUED)

61. Raju, G. V., and Bendazzi, A. "A Generalized Manufacturing Line Simulator System for Production Equipment and Manpower Planning." Proceedings of the Winter Simulation Conference, Washington, D.C. 1977.

62. Rappaport, Andy. "Instruments." EDN 27, no. 14 (1982): 106-133.

63. Rembold, U. "Justifying Process Control Computers in Discrete Parts Manufacturing Plants." Technical Paper MS73-508. Dearborn, Michigan: Society of Manufacturing Engineers, 1973.

64. Rembold, U.; Seth, M. K.; and Weinstein, J. S. Computers in Manufacturing. New York: Marcel Dekker, 1977.

65. Rose, Lawrence L. "TRM: A Resource Model for Network Processes." Fifteenth Annual Simulation Symposium, IEEE Computer Society, Tampa, Florida, December 1979. Silver Springs, Maryland: IEEE Press, 1980. Pp. 211-222.

66. Runner, J. S., and Leimkuhler, F. F. "CAMSAM: A Simulation Analysis Model for Computer-Aided Manufacturing Systems." Proceedings of the 1978 Summer Computer Simulation Conference, Newport Beach, California, July 1978.

67. Salomon, F. A., and Tafuri, D.A. "Emulation: A Useful Tool in the Development of Computer Systems." Fifteenth Annual Simulation Symposium, IEEE Computer Society, Tampa, Florida, December 1979. Silver Springs, Maryland: IEEE Press, 1980. Pp. 55-71.

68. Savas, Emanuel S. Computer Control of Industrial Processes. New York: McGraw-Hill, 1965.

69. Schagrin, E. F. "How Much Do Minicomputer Control Systems Cost?" Chemical Engineering 78, no.7 (March 1971): 103-109.

70. Schmidt, J. William, and Taylor, R. E. Simulation Analysis of Industrial Systems. Homewood, Illinois: R. D. Irwin, 1970.

71. Shaw, M. C. "Fundamentals of Wear." Annals of the CIRP 19 (1971): 533-543.

72. Solberg, J. J. "Can-Q User's Guide." Report No. 9, National Science Foundation Grant No. APR74 15256. West Lafayette, Indiana: Purdue University, June 1980.

REFERENCES CITED (CONTINUED)

73. "Sperry Univac Executive Asserts Manufacturing Control in Wrong Hands." Control Engineering 27, no. 7 (July 1980): 26.

74. Takeyama, H.; Doi, Y.; Mitsuoka, T.; and Sekiguchi, H. "Sensors of Tool Life for Optimization of Machining." Advances in Machining Tool Design and Research. Part 2 (1967): 191.

75. Takeyama, H.; Honda, T.; Sekiguchi, H.; Takada, K.; and Inoue, K. "Study on Automatic Programming for Numerical Control. Automatic Determination of Cutting Conditions in Longitudinal Rough Turning." Annals of the CIRP 19 (1971): 713-723.

76. Takeyama, H.; Sekiguchi, H.; and Takada, K. "One Approach for Optimizing Control in Metal Cutting." Annals of the CIRP 18 (1970): 345-351.

77. Vasilash, Gary S. "The Advent of Numerical Control: 1951-1959." Manufacturing Engineering 88 (1982): 143-172.

78. Vasilash, Gary S. "The Road to the Automatic Factory: 1970-1981." Manufacturing Engineering 88 (1982): 209-252.

79. Whetham, W. J. "Low Cost Adaptive Control Unit Manufacturing Methods." Technical Report AFML-TR-73-263. Wright-Patterson Air Force Base, Ohio: Air Force Materials Laboratory, November 1973.

80. Wrightman, E. J. Instrumentation in Process Control. Cleveland, Ohio: CRC Press, 1972.

81. Wu, S. M., and Ermer, D. S. "Maximum Profit as the Criterion in the Determination of the Optimum Cutting Conditions." Journal of Engineering for Industry 88 (November 1966): 435-440.

82. Wysk, Richard A.; Kimbler, Delbert L.; and Davis, Robert P. "Simulation of Adaptive Controlled Machining." Computers and Industrial Engineering 4 (1980): 69-74.

83. Yee, Kenneth W., and Blomquist, Donald S. An On-Line Method of Determining Tool Wear by Time-Domain Analysis. Technical Paper MR82-901. Dearborn, Michigan: Society of Manufacturing Engineers, 1982.

APPENDIX A

PROGRAM LISTING

```
      BLOCK DATA
      COMMON/CP10/TASK(8,5),NTAB(10),ISER(10),IBUS(10),IMACH(10),NMACH
      DATA TASK/4.000,8.000,10.000,12.000,18.000,20.000,21.000,22.000,
     *          2.000,2.000,01.000,01.000,01.000,02.000,01.000,01.000,
     *          4.120,2.095,04.300,05.050,02.180,04.390,02.2450,02.345,
     *          5.140,5.105,00.000,00.000,00.000,05.275,00.000,00.000,
     *          0.000,0.000,00.000,00.000,00.000,00.000,00.000,00.000/
      END
```

```
      DIMENSION NSET(10000)
      COMMON/SCOM1/ATRIB(100),DD(100),DDL(100),DTOW,II,MFA,MSTOP,NCLNR
     *,NCRDR,NPRNT,NNRUN,NNSET,NTAPE,SS(100),SSL(100),TNEXT,TNOW,XX(100)
      COMMON QSET(15000)
      EQUIVALENCE (NSET(1),QSET(1))
      NNSET=15000
      NCRDR=5
      NPRNT=6
      NTAPE=7
      CALL SLAM
      STOP
      END
```

```
      SUBROUTINE INTLC
      COMMON/SCOM1/ATRIB(100),DD(100),DDL(100),DTOW,II,MFA,MSTOP,NCLNR
     *,NCRDR,NPRNT,NNRUN,NNSET,NTAPE,SS(100),SSL(100),TNEXT,TNOW,XX(100)
      COMMON/CP10/TASK(8,5),NTAB(10),ISER(10),IBUS(10),IMACH(10),NMACH
      COMMON/UCOM2/MATRB,ISTA,NTRA,PRI(25),NTASK
C***    TRACE (NTRA) OUTPUT DEVICE
      NTRA=10
C***    STATE (ISTA) OUTPUT DEVICE
      ISTA=10
C***    SET THE NUMBER OF ATRIBUTES MATRB
      MATRB=17
C***    SET THE NUMBER OF MACHINES AND TRANSFER DEVICES IN THE CELL
      NMACH=6
C***    SET THE INITIAL PART EVENT FOR COMP20
      ATRIB(2)=1.0
      ATRIB(3)=20
      CALL SCHDL(20,0.0,ATRIB)
      RETURN
      END
```

```
C*************************************************************************************
C*
C*         COMPUTER AND DEVICE INTERFACE EVENTS
C*
C*************************************************************************************
      SUBROUTINE EVENT(IX)
      COMMON/SCOM1/ATRIB(100),DD(100),DDL(100),DTOW,II,MFA,MSTOP,NCLNR
     *,NCRDR,NPRNT,NNRUN,NNSET,NTAPE,SS(100),SSL(100),TNEXT,TNOW,XX(100)
      COMMON/UCOM2/MATRB,ISTA,NTRA,PRI(25),ITRP(25),NTASK
      COMMON/CP10/TASK(8,5),NTAB(10),ISER(10),IBUS(10),IMACH(10),NMACH
      DIMENSION ATEMP(30)
C***   CHECK ATRIB(2) FOR COMPLETION AND COMMUNICATION ROUTINE
25    IF(ATRIB(2).GE.1.0) GO TO 50
C***   COMUNICATION TO ANOTHER DEVICE
C***   SET ATRIB(2) TO CURRENT DEVICE
      ATRIB(2)=IX
C***   SET COMPUTER TO IDLE
      XX(IX)=0.0
C***    CHECK FOR PROCESSOR EXECUTION COMPLETION
      IF(ATRIB(4).EQ.0.0) GO TO 30
C***   RELEASE SERVICE COMMAND INTO SYSTEM
      CALL ENTER(IX,ATRIB)
C***   CHECK FOR OTHER REQUESTS
30    IF(NNQ(IX).LE.0) RETURN
      CALL RMOVE(1,IX,ATRIB)
C***     CHECK FOR REMAINING PROCESS TIME
      IF(ATRIB(2).LT.1.0) GO TO 25
      IF(ATRIB(2).GE.1.0)GO TO 70
      IF(ATRIB(4).LE.0.0) GO TO 25
C***   SCHEDULE EVENT COMPLETION WITH REMAINING PROCESSING TIME
      CALL SCHDL(IX,ATRIB(MATRB),ATRIB)
      RETURN
C***   IF PROCESSOR IS NOT BUSY SELECT SERVICE ROUTINE
50    IF(XX(IX).LE.0.0) GO TO 70
C***   CHECK INTERRUPT AND REMOVE CURRENT SERVICE ROUTINE
      IF((ATRIB(1).LE.PRI(IX)).OR.(ATRIB(1).LE.5.0)) GO TO 2000
C***   CHECK FOR INTERRUPT PRIORITY
      IRANK=NFIND(1,NCLNR,8,0,ITRP(IX),0.0)
C***   REMOVE LOW PRIORITY EVENT FROM THE EVENT CALANDER
      CALL RMOVE(IRANK,NCLNR,ATEMP)
      ATEMP(MATRB)=ATEMP(MATRB)-TNOW
C***   PUT THE EVENT IN THE COMPUTERS FILE
      ATEMP(2)=0.0
      CALL FILEM(IX,ATEMP)
C***  SET PROCESSOR IX TO BUSY
70    XX(IX)=1.0
C***   SET THE PRIORITY OF THE PROCESS ROUTINE
      PRI(IX)=ATRIB(1)
      ITRP(IX)=ATRIB(8)
C***  SELECT SERVICE COMPUTER
      GO TO(100,200,300,400,500,600,700,800,900,1000),IX
       CALL COMP20
      RETURN
100   CALL COMP1
      RETURN
```

```
 200    CALL COMP2
        RETURN
 300    CALL COMP3
        RETURN
 400    CALL COMP4
        RETURN
 500    CALL COMP5
        RETURN
 600    CALL COMP6
        RETURN
 700    CALL COMP7
        RETURN
 800    CALL COMP8
        RETURN
 900    CALL COMP9
        RETURN
 1000    CALL COMP10
        RETURN
 C***   PUT REQUEST IN FILE
 2000   CALL FILEM(IX,ATRIB)
        RETURN
        END
```

```
C*****************************************************************************
C*
C*      COMPUTER 10 PROGRAM SELECTION (CELL COMPUTER)
C*
C*****************************************************************************
      SUBROUTINE COMP10
      COMMON/SCOM1/ATRIB(100),DD(100),DDL(100),DTOW,II,MFA,MSTOP,NCLNR
     *,NCRDR,NPRNT,NNRUN,NNSET,NTAPE,SS(100),SSL(100),TNEXT,TNOW,XX(100)
      COMMON/UCOM2/MATRB,ISTA,NTRA,PRI(25),ITRP(25),NTASK
      COMMON/CP10/TASK(8,5),NTAB(10),ISER(10),IBUS(10),IMACH(10),NMACH
C***   SELECT COMPUTER SERVICE REQUEST ROUTINE BY ORIGIN DIVICE
      IDEV=ATRIB(2)
      GO TO(100,200,300,400,500,600,700,800,900,1000),IDEV
       CALL C10D20
      RETURN
100   CALL C10D01
      RETURN
200   CALL C10D02
      RETURN
300   CALL C10D02
      RETURN
400   CALL C10D02
      RETURN
500   CALL C10D02
      RETURN
600   CALL C10D06
      RETURN
700   CALL C10D07
      RETURN
800   CALL C10D08
      RETURN
900   CALL C10D09
      RETURN
1000   CALL C10D10
      RETURN
      END
```

```
C*************************************************************************************
C*
C* SERVICE PROGRAM FOR COMPUTER 10 FROM DEVICE 20 (CENTRAL COMPUTER)
C*
C*************************************************************************************
      SUBROUTINE C10D20
      COMMON/SCOM1/ATRIB(100),DD(100),DDL(100),DTOW,II,MFA,MSTOP,NCLNR
     *,NCRDR,NPRNT,NNRUN,NNSET,NTAPE,SS(100),SSL(100),TNEXT,TNOW,XX(100)
      COMMON/UCOM2/MATRB,ISTA,NTRA,PRI(25),ITRP(25),NTASK
      COMMON/CP10/TASK(8,5),NTAB(10),ISER(10),IBUS(10),IMACH(10),NMACH
      COMMON QSET(15000)
      IF(ATRIB(4).LE.-2.0) RETURN
      IQ=NNQ(1)
      IF(IQ.GT.15) GO TO 150
C***    SET ATRIB(2) TO SEND A MESSAGE
      ATRIB(2)=0.0
C***  CHECK MACHINE STATUS IN CELL AND PLACE TASK
C***  SET TASK POINTER TO IPT
      IPT=ATRIB(10)+10
      ITASK=ATRIB(IPT)
C***  LOCATE THE MACHINES FOR THE TASK
C***    IF CELL MACHINES CANNOT PERFORM TASK; SEND MESSAGE TO MAIN COMPUTER
      DO 100 IT=1,8
      IF(TASK(IT,1).EQ.ATRIB(IPT)) GO TO 200
100   CONTINUE
C***    SCHEDULE MESSAGE TO CENTRAL COMPUTER
150   ATRIB(4)=0.0
      ATRIB(2)=0.0
      ATRIB(3)=20.0
      CALL SCHDL(10,0.0015,ATRIB)
      RETURN
C***    IS THE TRANSFER INPUT DEVICE BUSY
200   IPRT=IPRT+1
      ATRIB(8)=IPRT
      IF(NTAB(1).LE.0.AND.ISER(1).LE.0.AND.ISER(NMACH+1).LE.0)GO TO 300
C***    STORE PART IN INPUT MEMORY QUEUE
250   CALL FILEM(1,ATRIB)
C***    SET NEXT PART FOR SERVICE IN MEMORY QUEUE
      ITEMP=MMFE(1)+8
      IMACH(1)=QSET(ITEMP)
C***    SCHDULE EXECUTION TIME
      ATRIB(4)=0.0
      CALL SYST(10)
      CALL SCHDL(10,0.0035,ATRIB)
      RETURN
C***    SEND TRANSFER MESSAGE TO INPUT DEVICE
C***    CHECK FOR ONE EMPTY POSITION ON THE TABLE
300   ISERM=0
      DO 325 I=1,NMACH
      IF(NTAB(I).GT.0) ISERM=ISERM+1
325   CONTINUE
C***    ONE EMPTY SPACE STORE THE PART IN THE QUEUE
      NTEMP=NMACH-1
      IF(ISERM.GE.NTEMP) GO TO 250
      ATRIB(4)=1.0
C***   SET PART ON THE INPUT TRANSFER DEVICE
      IMACH(1)=ATRIB(8)
      ATRIB(3)=1.0
      ISER(1)=1.0
      CALL SYST(101)
      CALL SCHDL(10,0.0020,ATRIB)
      RETURN
      END
```

```
C*************************************************************************************
C*
C*      SERVICE PROGRAM FOR COMPUTER 10 FROM DEVICE 2, 3 & 4 (MACHINES)
C*
C*************************************************************************************
      SUBROUTINE C10D02
      COMMON/SCOM1/ATRIB(100),DD(100),DDL(100),DTOW,II,MFA,MSTOP,NCLNR
     *,NCRDR,NPRNT,NNRUN,NNSET,NTAPE,SS(100),SSL(100),TNEXT,TNOW,XX(100)
      COMMON/UCOM2/MATRB,ISTA,NTRA,PRI(25),ITRP(25),NTASK
      COMMON/CP10/TASK(8,5),NTAB(10),ISER(10),IBUS(10),IMACH(10),NMACH
      ID=ATRIB(2)
      ATRIB(2)=0.0
C***    CHECK FOR TRANSFER COMPLETION
      IF(ATRIB(4).GT.0.0) GO TO 200
C***    SET THE SERVER TO IDLE
      ISER(ID)=0
C***    SINGLE TRANSFER WITHOUT MESSAGE TO START THE MACHINE
      IF(IMACH(ID).LE.0) GO TO 150
C***    TRANSFER TO MACHINE IS COMPLETE
      IMACH(ID)=ATRIB(8)
25    IBUS(ID)=1
      TIME=TIME+.0015
C***    SEND MESSAGE TO MACHINE
      ATRIB(4)=ABS(ATRIB(4))
      ATRIB(4)=EXPON(ATRIB(4),2)
      ATRIB(3)=ID
      CALL SCHDL(10,TIME,ATRIB)
C***    DO NOT REMOVE THE PART FROM CELL OUTPUT DEVICE
50    JMACH=NMACH-1
C***  REMOVE PART FROM CELL COMPUTER FILES
      DO 100 I=1,JMACH
      IN=NFIND(1,I,8,0,ATRIB(8),.1)
      IF(IN.LE.0) GO TO 100
      CALL ULINK(IN,I)
 100  CONTINUE
C***    TRANSFER TABLE STATUS
150   CALL SERCK(TIME)
      RETURN
C***   MACHINE HAS COMPLETED AN EVENT
200   IBUS(ID)=0
      IFLAG=1
      CALL MEMSTO(IFLAG,TIME)
C***    CHECK TRANSFER STATUS
      IF(IFLAG.GE.2) GO TO 25
      IF(ISER(NMACH+1).GT.0) GO TO 400
      TIME=0.001
C***    SET TEMP TO CURENT PART ON MACHINE
      ITEMP=IMACH(ID)
C***    SERVICE ROUTINE FOR POSSIBLE TRANSFER
      CALL TABSER(ID,TIME)
C***    NO TRANSFER; STORE PART INFO IN NMACH+1
      IF(ITEMP.EQ.IMACH(ID)) GO TO 400
300   ATRIB(4)=0.0
      CALL SCHDL(10,TIME,ATRIB)
      RETURN
400   ITEMP=NMACH+1
      CALL FILEM(ITEMP,ATRIB)
      ATRIB(4)=0.0
      CALL SCHDL(10,TIME,ATRIB)
      RETURN
      END
```

```
C*****************************************************************************
C*
C*      SERVICE PROGRAM FOR COMPUTER 10 FROM DEVICE 1 (INPUT)
C*
C*****************************************************************************
      SUBROUTINE C10D01
      COMMON/SCOM1/ATRIB(100),DD(100),DDL(100),DTOW,II,MFA,MSTOP,NCLNR
     *,NCRDR,NPRNT,NNRUN,NNSET,NTAPE,SS(100),SSL(100),TNEXT,TNOW,XX(100)
      COMMON/UCOM2/MATRB,ISTA,NTRA,PRI(25),ITRP(25),NTASK
      COMMON QSET(15000)
      COMMON/CP10/TASK(8,5),NTAB(10),ISER(10),IBUS(10),IMACH(10),NMACH
C***    SET EXECUTION TIME FOR ROUTINE
      TIME=0.001
      NTAB(1)=ATRIB(8)
      ISER(1)=0.0
C***    SET NEXT PART IN MEMORY QUEUE FOR TRANSFER
      IMACH(1)=0.0
      CALL SYST(1)
      I8=MMFE(1)
C***    CHECK FOR NEXT PART IN MEMORY QUEUE
      IF(I8.EQ.0) GO TO 100
      IMACH(1)=QSET(I8+8)
100   CALL MEMSTO(0,TIME)
C***    CHECK FOR OTHER TRANSFER OPERATIONS
      CALL SERCK(TIME)
      RETURN
      END
```

```
C*****************************************************************************
C*
C*      SERVICE PROGRAM FOR COMPUTER 10 FROM DEVICE 6 (OUTPUT)
C*
C*****************************************************************************
      SUBROUTINE C10D06
      COMMON/SCOM1/ATRIB(100),DD(100),DDL(100),DTOW,II,MFA,MSTOP,NCLNR
     *,NCRDR,NPRNT,NNRUN,NNSET,NTAPE,SS(100),SSL(100),TNEXT,TNOW,XX(100)
      COMMON/UCOM2/MATRB,ISTA,NTRA,PRI(25),ITRP(25),NTASK
      COMMON/CP10/TASK(8,5),NTAB(10),ISER(10),IBUS(10),IMACH(10),NMACH
      TIME=0.0015
C***    SET THE POSITION ON THE TABLE TO EMPTY
      IMACH(NMACH)=0
      ISER(NMACH)=0
      NTAB(NMACH)=0
      ATRIB(2)=0.0
C***    SEND THE COMPLETION MESSAGE TO MAIN COMPUTER
      ATRIB(3)=20.0
      ATRIB(4)=-1.0
      CALL SCHDL(10,TIME,ATRIB)
C***    CHECK FOR NEXT TABLE STEP
      CALL SERCK(TIME)
      RETURN
      END
```

```
C*****************************************************************************
C*
C*      SERVICE PROGRAM FOR COMPUTER 10 FROM DEVICE 7
C*
C*****************************************************************************
      SUBROUTINE C10D07
      COMMON/SCOM1/ATRIB(100),DD(100),DDL(100),DTOW,II,MFA,MSTOP,NCLNR
     *,NCRDR,NPRNT,NNRUN,NNSET,NTAPE,SS(100),SSL(100),TNEXT,TNOW,XX(100)
      COMMON/UCOM2/MATRB,ISTA,NTRA,PRI(25),ITRP(25),NTASK
      COMMON/CP10/TASK(8,5),NTAB(10),ISER(10),IBUS(10),IMACH(10),NMACH
      COMMON QSET(15000)
      DIMENSION IQ(20,10)
      CALL SYST(70)
C***    SET THE TRANSFER TABLE TO IDLE
      ISER(NMACH+1)=0
      TEMP1=NTAB(NMACH)
      TEMP2=NTAB(1)
C***    SERVICE INDICATOR TO IDLE
      ISER(NMACH+1)=0
C***    UPDATE TABLE AND CHECK TABLE STATUS
      TIME=0.001
      DO 100 I=1,NMACH
C***    UPDATE POSITION I ON THE TABLE
      NTAB(I)=TEMP1
      TEMP1=TEMP2
      TEMP2=NTAB(I+1)
      TIME=TIME+0.001
C***    CHECK DEVICE FOR TRANSFER
      IF(IBUS(I).GT.0) GO TO 100
C***    CALL TRANSFER SERVICE ROUTINE FOR TABLE STATUS AND POSSIBLE TRANSFER
      CALL TABSER(I,TIME)
100   CONTINUE
C***    SET THE CELL PROCESSOR TO CURRENT INTERRUPT PRIORITY
      ATRIB(1)=PRI(10)
      CALL SYST(71)
C***    CHECK FOR TRANSFER TABLE STATUS
      CALL SERCK(TIME)
      RETURN
C***    SCHEDULE COMPLETION FOR PROGRAM EXECUTION
200   TIME=TIME+0.001
      ATRIB(4)=0.0
      CALL SCHDL(10,TIME,ATRIB)
      RETURN
      END
```

```
C*****************************************************************************
C*
C*       SERVICE PROGRAM FOR CHECKING THE TRANSFER DEVICE
C*
C*****************************************************************************
      SUBROUTINE SERCK(XTIME)
      COMMON/SCOM1/ATRIB(100),DD(100),DDL(100),DTOW,II,MFA,MSTOP,NCLNR
     *,NCRDR,NPRNT,NNRUN,NNSET,NTAPE,SS(100),SSL(100),TNEXT,TNOW,XX(100)
      COMMON/UCOM2/MATRB,ISTA,NTRA,PRI(25),ITRP(25),NTASK
      COMMON/CP10/TASK(8,5),NTAB(10),ISER(10),IBUS(10),IMACH(10),NMACH
C***    THIS SUBROUTINE CHECKS FOR TRANSFER TABLE STATUS AND STEPS TABLE
      NCK=0
C***   NO OF SERVERS
      NSERV=NMACH+1
      ATRIB(2)=0.0
      ATRIB(4)=0.0
      TIME=0.001+XTIME
      DO 100 I=1,NSERV
      XTIME=XTIME+0.0015
C***    CHECK FOR TRANSFER SERVICE COMPLETIONS
      IF(ISER(I).GT.0) GO TO 200
      NCK=NCK+NTAB(I)
100   CONTINUE
```

```
C***     CHECK FOR PARTS ON THE TABLE
      IF(NCK.LE.0) GO TO 200
C***     SCHEDLUE COMMAND FOR ONE STEP OF THE TABLE
      ATRIB(4)=1.0
      ATRIB(3)=NSERV
C***     SET SERVICE INDICATOR TO BUSY
      ISER(NSERV)=1
      CALL SCHDL(10,XTIME,ATRIB)
      RETURN
C***     SCHEDULE EXECUTION COMPLETION FOR CELL COMPUTER
200   CALL SCHDL(10,XTIME,ATRIB)
      RETURN
      END
```

```
C*************************************************************************************
C*
C*     STORE THE TASK IN THE COMPUTER MEMORY FOR EACH MACHINE
C*
C*************************************************************************************
      SUBROUTINE MEMSTO(NFLAG,XTIME)
      COMMON/SCOM1/ATRIB(100),DD(100),DDL(100),DTOW,II,MFA,MSTOP,NCLNR
     *,NCRDR,NPRNT,NNRUN,NNSET,NTAPE,SS(100),SSL(100),TNEXT,TNOW,XX(100)
      COMMON/UCOM2/MATRB,ISTA,NTRA,PRI(25),ITRP(25),NTASK
      COMMON/CP10/TASK(8,5),NTAB(10),ISER(10),IBUS(10),IMACH(10),NMACH
C***     THIS SUBOUTINE STORES TASKS IN MEMORY QUEUES FOR EACH DEVICE
C***     SET THE TASK POINTER (IPT)
      IPT=ATRIB(10)+10
      IF(NFLAG.GE.1) IPT=IPT+1
C***     IS THE PART FINISHED FOR CURRENT CELL?
      IF(ATRIB(IPT).EQ.0.0) GO TO 200
C***     PLACE TASK IN MEMORY QUEUES THAT CAN PERFORM THE TASK
      DO 50 JTASK=1,8
      XTIME=XTIME+0.0015
      IF(ATRIB(IPT).EQ.TASK(JTASK,1)) GO TO 75
50    CONTINUE
75    KTASK=TASK(JTASK,2)+2
      DO 100 J=3,KTASK
C***     LOOP EXECUTION TIME
      XTIME=XTIME+.0070
      JMACH=TASK(JTASK,J)
C***     SET THE ESTIMATED PROCESS TIME
      ATRIB(4)=(TASK(JTASK,J)-JMACH)*1000
C***     CHECK IF THE NEXT TASK CAN BE DONE ON SAME MACHINE
      IF(IMACH(JMACH).EQ.IFIX(ATRIB(8))) GO TO 150
C***     FILE IN THE MEMORY QUEUE FOR MACHINE (JMACH)
      IF(NFLAG.LE.0) CALL FILEM(JMACH,ATRIB)
100   CONTINUE
      RETURN
150   NFLAG=2
      ATRIB(10)=ATRIB(10)+1
      RETURN
C***     PUT PART IN THE CELL OUTPUT MEMORY QUEUE
200   IF(NFLAG.LE.0) CALL FILEM(NMACH,ATRIB)
C***     EXECUTION TIME
      XTIME=XTIME+0.001
      RETURN
      END
```

```
C*****************************************************************************
C*
C*      SERVICE ROUTINE FOR DETERMINING TRANSFER EVENTS
C*
C*****************************************************************************
      SUBROUTINE TABSER(I,XTIME)
      COMMON/SCOM1/ATRIB(100),DD(100),DDL(100),DTOW,II,MFA,MSTOP,NCLNR
     *,NCRDR,NPRNT,NNRUN,NNSET,NTAPE,SS(100),SSL(100),TNEXT,TNOW,XX(100)
      COMMON/UCOM2/MATRB,ISTA,NTRA,PRI(25),ITRP(25),NTASK
      COMMON/CP10/TASK(8,5),NTAB(10),ISER(10),IBUS(10),IMACH(10),NMACH
      LOGICAL TABL,MACHL
      MACHL=.FALSE.
      TABL=.FALSE.
      IFIND=0.0
C***    MACHINE BUSY; RETURN
      IF(IBUS(I).GT.0) RETURN
C***    SET TABLE AND MACHINE STATUS TRUE FOR HAVING PART
      IF(NTAB(I).GT.0) TABL=.TRUE.
      IF(IMACH(I).GT.0) MACHL=.TRUE.
C***    NO PARTS; RETURN
      IF((.NOT.TABL).AND.(.NOT.MACHL)) RETURN
C***    SERVICE THE INPUT QUEUE
      IF(I.NE.1) GO TO 100
      IFIND=1
      ITEMP1=NNQ(1)
      IF(ITEMP1.GT.0.AND.NTAB(1).LE.0) GO TO 400
      RETURN
C***    PART ON MACHINE BUT NOT ON TABLE; GO TO 150
100   IF((.NOT.TABL).AND.MACHL) GO TO 150
      TEMP2=NTAB(I)
170   IFIND=NFIND(1,I,8,0,TEMP2,0.01)
C***    PART IS CONTAINED IN MEMORY FOR MACHINE; RETURN
      IF(IFIND.LE.0) RETURN
      IF(.NOT.MACHL) GO TO 400
150   ITEMP=NMACH+1
C***   NO TABLE PART BUT PART ON THE MACHINE
      XTIME=XTIME+0.001
      TEMP=NNQ(ITEMP)
C***   IF THE PART ON MACHINE HAS JUST FINISHED; GO TO 200
      IF(TEMP.LE.0.0) GO TO 200
      TEMP=IMACH(I)
C***    FIND PART ON MACHINE AWAITING SERVICE
      IRANK=NFIND(1,ITEMP,8,0,TEMP,0.01)
      IF(IRANK.LE.0) GO TO 200
C***  REMOVE PART FROM THE INTERMEDIATE STORAGE FILE
      CALL RMOVE(IRANK,ITEMP,ATRIB)
C***    STORE NEXT TASK IN MACHINE MEMORIES
200   ATRIB(10)=ATRIB(10)+1
      XTIME=XTIME+0.015
C***    IF TABLE POSITION DOES NOT HAVE A PART; SCHEDULE TRANSFER
      CALL MEMSTO(0,TIME)
C***    REMOVE NEXT PART FORM MEMORY AND SCHEDULE TRANSFER
400   IF((IFIND.GT.0).OR.TABL.OR.(I.EQ.1)) CALL RMOVE (IFIND,I,ATRIB)
      TEMP1=IMACH(I)
      IMACH(I)=NTAB(I)
      NTAB(I)=TEMP1
      ISER(I)=1
      ATRIB(4)=ATRIB(4)*(-1.0)
500   ATRIB(2)=0.0
      ATRIB(3)=I
      XTIME=XTIME+0.002
      CALL SCHDL(10,XTIME,ATRIB)
      RETURN
      END
```

```
      SUBROUTINE COMP20
      COMMON/SCOM1/ATRIB(100),DD(100),DDL(100),DTOW,II,MFA,MSTOP,NCLNR
     *,NCRDR,NPRNT,NNRUN,NNSET,NTAPE,SS(100),SSL(100),TNEXT,TNOW,XX(100)
      COMMON/CP10/TASK(8,5),NTAB(10),ISER(10),IBUS(10),IMACH(10),NMACH
      COMMON/UCOM2/MATRB,ISTA,NTRA,PRI(25),ITRP(25),NTASK
      DIMENSION IPART(25,5)
      DATA IPART/01,01,01,01,01,01,01,01,00,00,00,00,00,00,00,
     *          00,00,00,00,00,00,00,00,00,00,
     *             18,08,18,08,12,12,18,00,00,00,00,00,00,00,00,
     *          00,00,00,00,00,00,00,00,00,00,
     *             22,21,22,21,21,00,20,00,00,00,00,00,00,00,00,
     *          00,00,00,00,00,00,00,00,00,00,
     *             20,12,10,04,00,00,00,00,00,00,00,00,00,00,00,
     *          00,00,00,00,00,00,00,00,00,00,
     *             00,00,00,12,00,00,00,00,00,00,00,00,00,00,00,
     *          00,00,00,00,00,00,00,00,00,00/
C***     INCREMENT THE PART NUMBER
C***     SET CONTROL ATRIBUTES
      ATRIB(8)=IPRT
      ATRIB(1)=6.0
      ATRIB(2)=0.0
      ATRIB(3)=10.0
      ATRIB(4)=1.0
      ATRIB(6)=TNOW
C***     SAMPLE PART TYPE FROM UNIFORM DISTRIBUTION
      ICHO=UNFRM(1.0,14.0,1)
      WRITE(ISTA,4000)IPRT,ICHO,TNOW
4000  FORMAT(' PART NO.',I3,' IS PART TASK',I3,' TIME=',F10.3)
      ATRIB(15)=ICHO
      DO 100 I=1,5
C***     SET THE TASK NUMBERS FOR PARTS
      ATRIB(I+9)=IPART(ICHO,I)
      WRITE(NTRA,4001)I,ATRIB(I+9)
4001  FORMAT(' THE PART',I3,' TASK IS ',F10.2)
100   CONTINUE
C***     SEND MESSAGE TO CELL COMPUTER 10
      CALL SCHDL(20,0.01,ATRIB)
C***     SCHEDULE NEXT PART INTO SYSTEM
      ATRIB(2)=20.0
      RATEIN=EXPON(210.0,3)
      CALL SCHDL(20,RATEIN,ATRIB)
      RETURN
      END


      SUBROUTINE SYST(IR)
      COMMON/SCOM1/ATRIB(100),DD(100),DDL(100),DTOW,II,MFA,MSTOP,NCLNR
     *,NCRDR,NPRNT,NNRUN,NNSET,NTAPE,SS(100),SSL(100),TNEXT,TNOW,XX(100)
      COMMON/UCOM2/MATRB,ISTA,NTRA,PRI(25),ITRP(25),NTASK
      COMMON/CP10/TASK(8,5),NTAB(10),ISER(10),IBUS(10),IMACH(10),NMACH
      COMMON QSET(15000)
      DIMENSION IQ(20,10)
      WRITE(ISTA,4006)IR,TNOW
4006  FORMAT(' STATE DURING ROUTINE',I3,'; TIME =',F10.0)
      WRITE(ISTA,4002)(IMACH(I1),I1=1,7)
4002  FORMAT(/,' PARTS ON MACHINE  ',715)
      WRITE(ISTA,4003)(IBUS(I1),I1=1,7)
4003  FORMAT(' MACHINE STATUS     ',715)
      WRITE(ISTA,4004)(ISER(I1),I1=1,7)
4004  FORMAT(' TRANSFER STATUS    ',715)
      WRITE(ISTA,4005)(NTAB(I1),I1=1,6)
4005  FORMAT(' TABLE PARTS        ',615)
      DO 500 I=1,7
      INUM=NNQ(I)
      IF(INUM.LE.0) GO TO 625
      DO 600 J=1,INUM
      IPT=LOCAT(J,I)+8
      IQ(J,I)=QSET(IPT)
600   CONTINUE
```

```
625     INUM=INUM+1
        DO 650 K=INUM,5
        IQ(K,I)=0
650     CONTINUE
500     CONTINUE
        WRITE(ISTA,4011)
4011    FORMAT(/,'PARTS IN THE MEMORY QUEUES')
        DO 700 JK=1,5
        WRITE(ISTA,4010)(IQ(JK,M),M=1,7)
4010    FORMAT(19X,715)
700     CONTINUE
        RETURN
        END


GEN,SCOTT,SNOP&NAM: 3 MACHINES,11/17/82,10,NO,NO,YES,YES,YES;
LIMITS,20,20,250;
;PRORITY/1,LVF(20);
PRIORITY/2,FIFO;
PRIORITY/3,FIFO;
PRIORITY/4,FIFO;
PRIORITY/5,FIFO;
PRIORITY/6,FIFO;
PRIORITY/7,FIFO;
PRIORITY/8,FIFO;
PRIORITY/9,FIFO;
PRIORITY/10,FIFO;
PRIORITY/20,FIFO;
TIMST,XX(10),CP10 UTIL;
TIMST,XX(20),CP80 UTIL;
NETWORK;
        ENTER,20,1;
PR10    GOON;
        ACT,ATRIB(7);
        EVENT,10;
        TERM;
        ENTER,10,1;
        ACT,0.01,ATRIB(3).LE.1.0,PR01;
        ACT,0.01,ATRIB(3).LE.2.0,PR02;
        ACT,0.01,ATRIB(3).LE.3.0,PR03;
        ACT,0.01,ATRIB(3).LE.4.0,PR04;
        ACT,0.01,ATRIB(3).LE.5.0,PR05;
        ACT,0.01,ATRIB(3).LE.6.0,PR06;
        ACT,0.01,ATRIB(3).LE.7.0,PR07;
        ACT,0.01,ATRIB(3).LE.20.0,PR20;
PR01    ASSIGN,ATRIB(2)=1.,ATRIB(3)=10.,ATRIB(7)=ATRIB(5)/200.0,
        ATRIB(4)=ATRIB(4)*-1.0,1;
        ACT,ATRIB(7);
TR01    GOON,1;
        ACT/1,EXPON(30.0,1),,PR10;
        TERM;
PR02    ASSIGN,ATRIB(2)=2.,ATRIB(3)=10.,ATRIB(7)=ATRIB(5)/200.0,1;
        ACT,ATRIB(7);
MH02    GOON,1;
        ACT,RNORM(15.0,5.0),ATRIB(4).LE.-1.0,PR10;
        ACT/2,ATRIB(4),,PR10;
        TERM;
PR03    ASSIGN,ATRIB(2)=3.,ATRIB(3)=10.,ATRIB(7)=ATRIB(5)/200.0,1;
        ACT,ATRIB(7);
MH03    GOON,1;
        ACT,RNORM(17.0,3.0),ATRIB(4).LE.-1.0,PR10;
        ACT/3,ATRIB(4),,PR10;
        TERM;
PR04    ASSIGN,ATRIB(2)=4.0,ATRIB(3)=10.0,ATRIB(7)=ATRIB(5)/200.0,1;
        ACT,ATRIB(7);
MH04    GOON,1;
        ACT,RNORM(22.0,9.0,1),ATRIB(4).LE.-1.0,PR10;
        ACT/4,ATRIB(4),,PR10;
        TERM;
PR05    ASSIGN,ATRIB(2)=5.0,ATRIB(3)=10.0,ATRIB(7)=ATRIB(5)/200.0,1;
```

```
        ACT,ATRIB(7);
MH05    GOON,1;
        ACT,RNORM(12.0,1.0,1),ATRIB(4).LE.-1.0,PR10;
        ACT/5,ATRIB(4),,PR10;
        TERM;
PR06    ASSIGN,ATRIB(2)=6.0,ATRIB(3)=10.0,ATRIB(7)=ATRIB(5)/200.0,1;
        ACT,ATRIB(7);
MH06    GOON,1;
        ACT/6,RNORM(5.0,0.5);
        COLCT,INT(6),TIME IN CELL;
        COLCT,BET,TIME BET DEPT;
        ACT,,,PR10;
        TERM;
PR07    ASSIGN,ATRIB(2)=7.,ATRIB(3)=10.,ATRIB(7)=ATRIB(5)/200.0,1;
        ACT,ATRIB(7);
TR07    GOON,1;
        ACT/7,UNFRM(10.0,12.0),,PR10;
        TERM;
PR20    ASSIGN,ATRIB(2)=20.0,ATRIB(3)=10.0,ATRIB(7)=ATRIB(5)/200.0,1;
        ACT,ATRIB(7);
        TERM;
        ENDNETWORK;
INTLC,XX(100)=10.0;
INIT,0,11400;
FIN:
```

APPENDIX B

CNC LATHE CONTINUOUS MODEL

An adaptive control system for a CNC turning operation is simulated as a continuous model. The adaptive control system maintains a constant cutting force. The SLAM continuous simulation is based on the dynamic relationships of CNC turning operations. The SLAM graphics output is demonstrated with a sample turning operation. This output format can be compared directly with measured data.

CNC Control System

A pulse reference system is used to control the feed rate servomotor on the CNC lathe [44], as shown in Figure B.1. Each pulse moves the tool feed one basic length unit (BLU). The frequency of the pulses is proportional to feed rate. The computer determines the reference-pulse rate, based on the error between the measured cutting force and the reference cutting force. Two control loops are used to maintain a constant cutting force (Figure B.2).

The inner feedback loop regulates feed rate. An up-down counter determines the pulse number difference (phase difference) between the reference pulse and the measured pulse from the encoder. This up-down counter actually integrates the error of the pulse frequency. The DAC converts the pulse count into voltage drive for the motor. Any position error between the two inputs produces changes in motor voltage, thereby reducing or increasing speed. The drive motor is modeled as a time-constant transfer equation with τ equal to

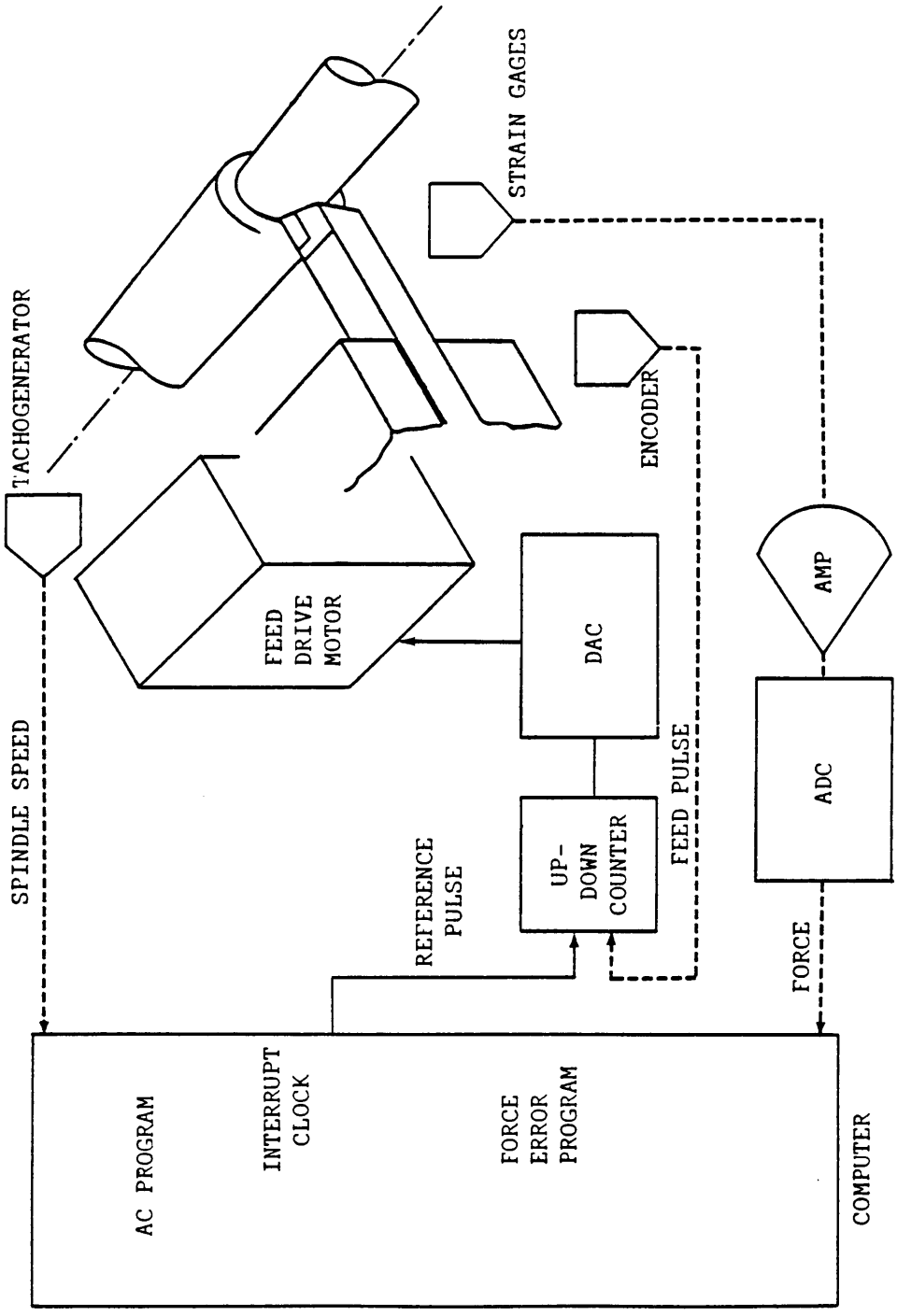

Figure B.1. CNC Lathe Control System

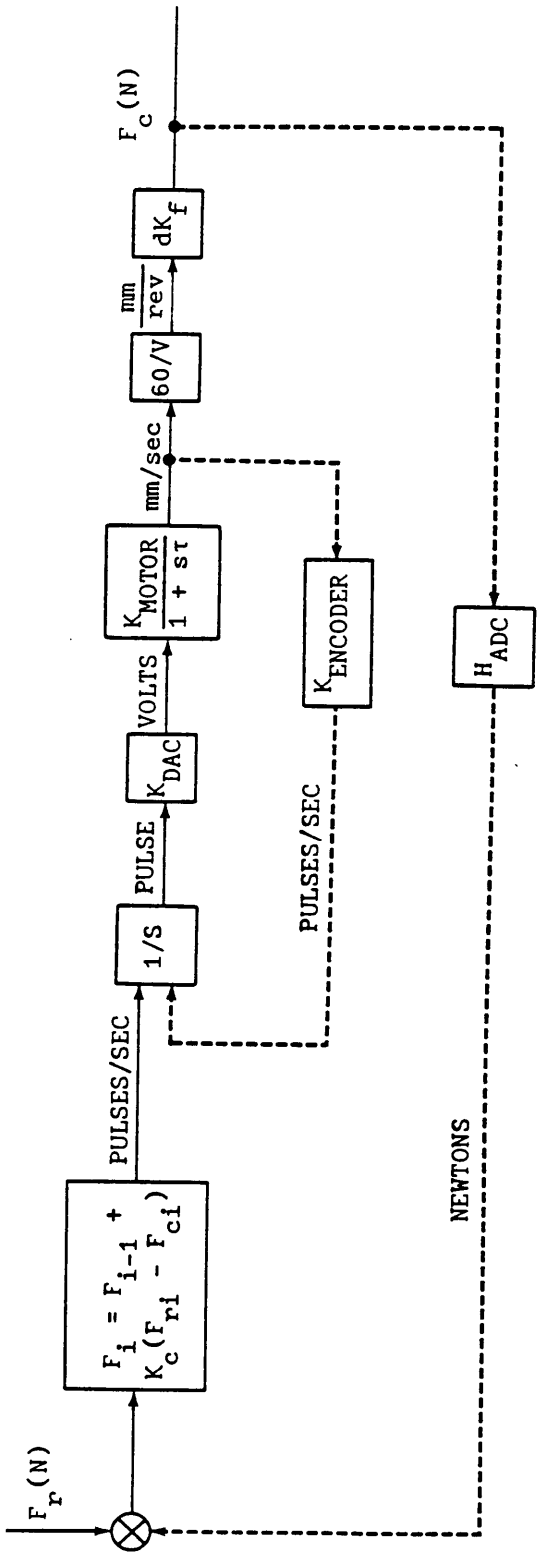

Figure B.2. CNC Lathe Control System Transfer Diagram

12.5 milliseconds. The BLU for this example is 0.01 millimeter. Pulse frequency is proportional to speed [44].

The loop with the up-down counter is an integrator error control loop. By multiplying the gains and LaPlace transforms of the transfer blocks for each device, the closed loop transform becomes:

$$G(s) = \frac{K}{\tau s^2 + s + K}$$

where $K = K_{MOTOR} * K_{DAC} * K_{ENCODER}$

This is a second-order servocontrol system.

The outside control loop maintains a constant cutting force by regulating the feed rate reference pulse. Cutting force gain is based on the material characteristics. Note this gain can also be a random variable based on a Markov process. The actual force is determined by multiplying cutting force gain (K_f) by cutting speed, feed rate, and depth of cut. Cutting force from the actual system is summed with the reference force to determine the error. This error is added to the force calculated in the previous time interval as follows:

$$F_i = F_{i-1} + K_c(F_{ri} - F_{ci})$$

where:

i - sampling interval

K_c - controller gain

F_r - reference force

F_c - cutting force

Thus, the force control loop is also an integral controller. This calculation is usually performed by a microprocessor.

At precise time intervals (every 0.01 second), the processor sends the pulse rate to a pulse generator. This rate is proportional to the power requirements determined in the microprocessor's logic routine. This proportion can be represented by a gain K_p. This pulse rate is used to change the voltage to the motor. The servomotor control loop maintains the speed. This outside control loop gain is affected by the depth of cut and spindle speed. Reducing the depth of cut or spindle speed increases the gain in the system, causing instability. The entire system is simulated with the model structure for a CNC lathe.

SLAM CNC Lathe Model

The inverse transform of the feed control transfer function G(s) is performed, assuming all initial input states are zero. Thus, the relationship for the speed control transfer block is:

$$y''(t) = (((x(t) - y(t)) * K) - y'(t))/\tau$$

where:

y(t) = feed speed (mm/second)

x(t) = reference pulse (pulses/second)

K = feed control loop gain (mm/pulse)

Once this transform is made, the entire system can be modeled in the continuous portion of SLAM. Figure B.3 shows the continuous model, along with the control gains and other variables. The above equation determines the feed rate which varies with the force SS(7). SS(6) is the cutting force constant, which varies ±2.5 percent of the cutting force gain K_f. All values were obtained for P-25 coated carbide inserts with SAE 1045 0.55 percent carbon steel [44].

```
C***     SET THE CONTROL LOOP GAIN CONSTANTS
C***    XX(51):  REFERENCE CUTTING FORCE
C***    XX(52):  CONTROL SIGNAL GAIN (KF)
C***    XX(53):  FORCE ERROR SIGNAL GAIN (KC)
C***    XX(54):  MOTOR CONTROL SYSTEM GAIN (K)
C***    XX(55):  BASIC LENGTH UNIT (MM)
C***    XX(56): CURRENT COMPUTER CUTTING FORCE
C***    XX(57): PREVIOUS SAMPLE CUTTING FORCE
C***    XX(59): TURNING STOCK LENGTH (MM)
       XX(51)=1500.0
       XX(52)=0.0333333
       XX(53)=0.5
       XX(54)=42.0
       XX(55)=0.01
       XX(56)=1500.0
       XX(57)=1500.0
       XX(59)=25.0
C***     SYSTEM STATE VARIABLES
C***    SS(1):  FEED VELOCITY (MM/S)
C***    SS(2):  FEED ACCELERATION (MM/S**2)
C***    SS(3):  REFERENCE CONTROL PULSE TRAIN (PULSES/S)
C***    SS(4):  SERVO TIME CONSTANT (S)
C***    SS(5):  SPINDLE SPEED (RPM)
C***    SS(6):  CUTTING FORCE GAIN (KS) (NEWTON-REV/MM**2)
C***    SS(7):  CUTTING FORCE (NEWTONS)
C***    SS(8):  FORCE CONTROL SIGNAL GAIN (PLUSES-REV/NEWTON-S)
C***    SS(9):  DEPTH OF CUT (MM)
C***    SS(10): TOOL FEED (MM/REV)
C***   SS(11):  TOTAL FEED DISTANCE
       SS(1)=0.0
       SS(2)=0.0
       SS(3)=0.0
       SS(4)=0.0125
       SS(5)=600.0
       SS(6)=3000.0
       SS(7)=0.0
       SS(8)=0.0
       SS(9)=0.0
       SS(10)=0.0
       RETURN
       END
       SUBROUTINE EVENT(IX)
       COMMON/SCOM1/ATRIB(100),DD(100),DDL(100),DTOW,II,MFA,MSTOP,NCLNR
      *,NCRDR,NPRNT,NNRUN,NNSET,NTAPE,SS(100),SSL(100),TNEXT,TNOW,XX(100)
       COMMON QSET(15000)
       COMMON/UCOM1/TMAX(10),TMIN(10),RATE2,RATE3,RATE4
       GO TO (1,2,3,4,5,6,7,8),IX
1      SS(9)=2.0
       RETURN
2      SS(9)=4.0
       RETURN
3      SS(9)=2.5
```

Figure B.3. CNC Lathe Continuous Program Listing

```
      RETURN
4     SS(9)=6.0
5     SS(9)=1.0
      RETURN
6     SS(9)=2.0
      RETURN
7     SS(9)=0.0
      RETURN
C***     PROCESSOR FORCE INTEGRATION ROUTINE
8     CALL SCHDL(8,0.1,ATRIB)
      IF(SS(9).LE.0.0) GO TO 200
      XX(56)=XX(57)+XX(53)*(XX(51)-SS(7))
      XX(57)=XX(56)
      RETURN
200   XX(56)=1500.0
      RETURN
      END
      SUBROUTINE STATE
      COMMON/SCOM1/ATRIB(100),DD(100),DDL(100),DTOW,II,MFA,MSTOP,NCLNR
     *,NCRDR,NPRNT,NNRUN,NNSET,NTAPE,SS(100),SSL(100),TNEXT,TNOW,XX(100)
      COMMON/UCOM1/TMAX(10),TMIN(10),RATE2,RATE3,RATE4
      SS(10)=SS(1)*XX(55)*(60.0/SS(5))
      SS(3)=XX(56)*XX(52)*SS(5)/60.0
C***    PROCESSOR FORCE GAIN
C***    QUADRATIC CONTROL LOOP
      DD(1)=SS(2)
      DD(2)=(XX(54)*(SS(3)-SS(1))-DD(1))/SS(4)
C***   CUTTING FORCE GAIN
      SS(7)=SS(9)*SS(6)*SS(10)*UNFRM(0.98,1.02,1)
C***    ENCODER POSITION MEASUREMENT
      DD(11)=SS(1)*XX(55)
      RETURN
      END
GEN,H. A. SCOTT,DYNAMIC,10/16/82,1;
LIMITS,4,4,100;
CONTINUOUS,11,11,,0.1,,N;
RECORD,TNOW,TIME,0,P,0.5,,,NO;
VAR,SS(10),Y,VELOCITY,0.0,2.0;
VAR,SS(9),T,DEPTH,0.0,10.0;
VAR,SS(7),I,FORCE,0.0,2000.0;
VAR,SS(3),S,SENSOR,0.0,2000.0;
SEVNT,1,SS(11),XP,4.0,0.01;
SEVNT,2,SS(11),XP,25.0,0.01;
SEVNT,3,SS(11),XP,60.0,0.01;
SEVNT,1,SS(11),XP,79.0,0.01;
SEVNT,7,SS(11),XP,90.0,0.01;
SEVNT,5,SS(11),XP,160.0,0.01;
SEVNT,1,SS(11),XP,180.0,0.01;
SEVNT,6,SS(11),XP,240.0,0.01;
SEVNT,10,SS(11),XP,300.0,0.01;
INIT,0,1000.0;
FIN;
```

Figure B.3. CNC Lathe Continuous Program Listing (Continued)

SS(11) is the current position of the tool in relationship to the starting position. The SLAM statement, DD(11) = SS(1) * XX(55), integrates the actual number of pulses from the system over time, multiplied by the BLU. This simulates an encoder and provides actual tool position from the initial reference point.

SS(3) is the difference between the two forces multiplied by the error control gain. This is the control reference pulse over time. Every 0.1 second, an event 8 is scheduled to determine the difference between the reference force and actual force. The difference is multiplied by the error gain X(53) and added to the force from the previous sampling period XX(57). If the force is zero, a 1500-Newton force signal is sent, producing a 0.5 millimeter per second feed rate. Thus, a force-controlled CNC lathe is simulated in only 15 lines of SLAM source program code. Constant spindle speed is assumed.

Output from this control system is shown in Figure B.4. At 600 rpm, a cutting sequence is shown that is almost identical to the results provided in reference [44]. The change in cutting depth is performed in the discrete program subroutine C10D05. When S(11) reaches a specified distance, a call is made to COMP05, which changes cutting depth according to the part program array. SS(9) is changed to the required depth. Thus, the actual lathe program sequence can be tested with this simulator.

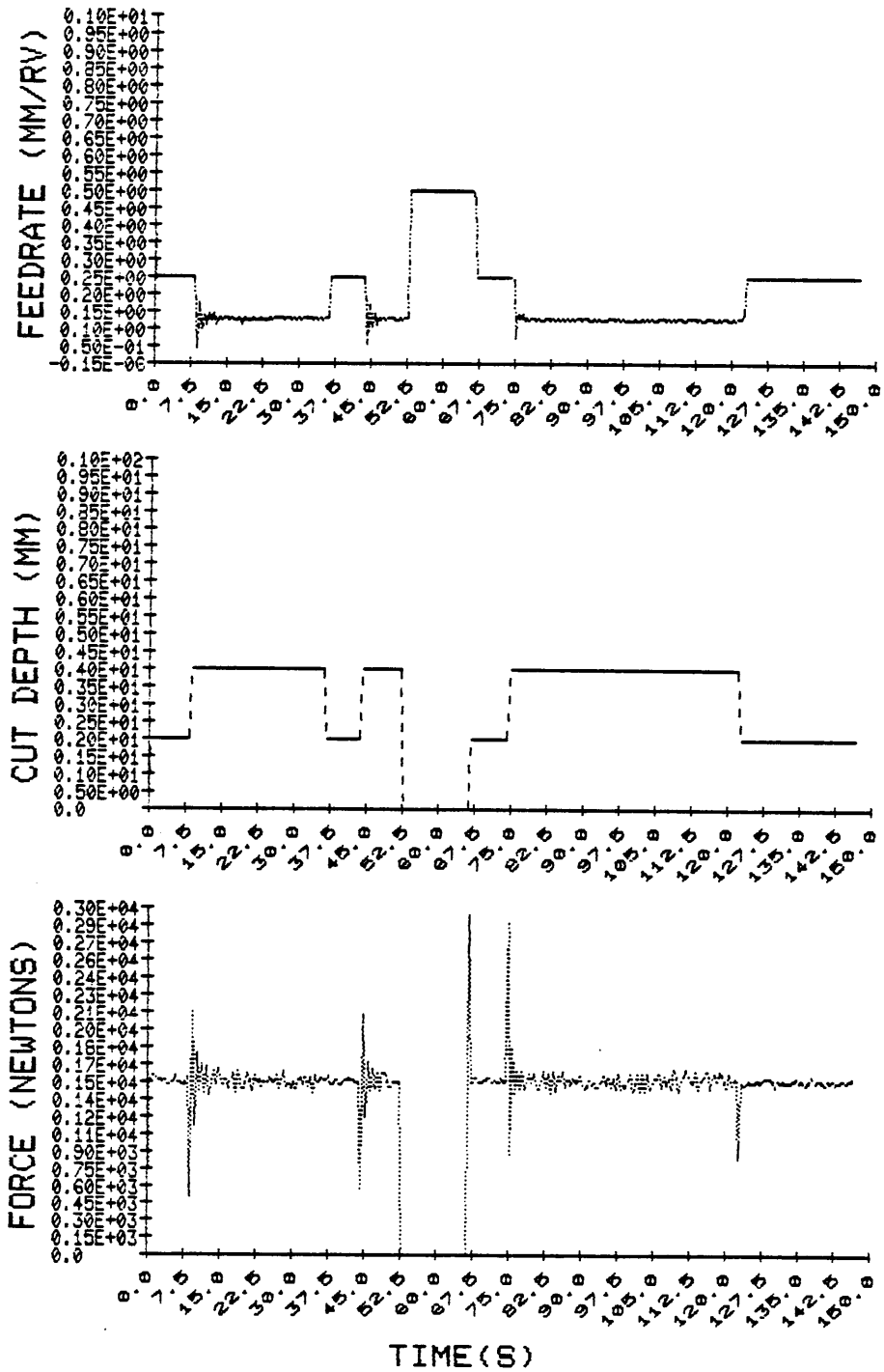

Figure B.4. Cutting Trace (600 rpm)

Summary

Tool breakage can also be addressed by the model. Figure B.5 shows a run in which depth of cut increased 6 millimeters almost instantaneously. Instability in the system indicates tool failure. The control system could not respond in time to this sudden force. Thus, NC lathe programs can be tested before implementation on an actual machine to avoid tool breakage.

Other machining processes can be analyzed with the SLAM continuous model. This capability enables tool designers and control engineers to experiment with advanced control and tool designs before building the prototype system. In addition, this continuous model can be integrated directly into the entire manufacturing process simulation with an interface module. Thus, machine performance can be analyzed with other processes in a simulated system.

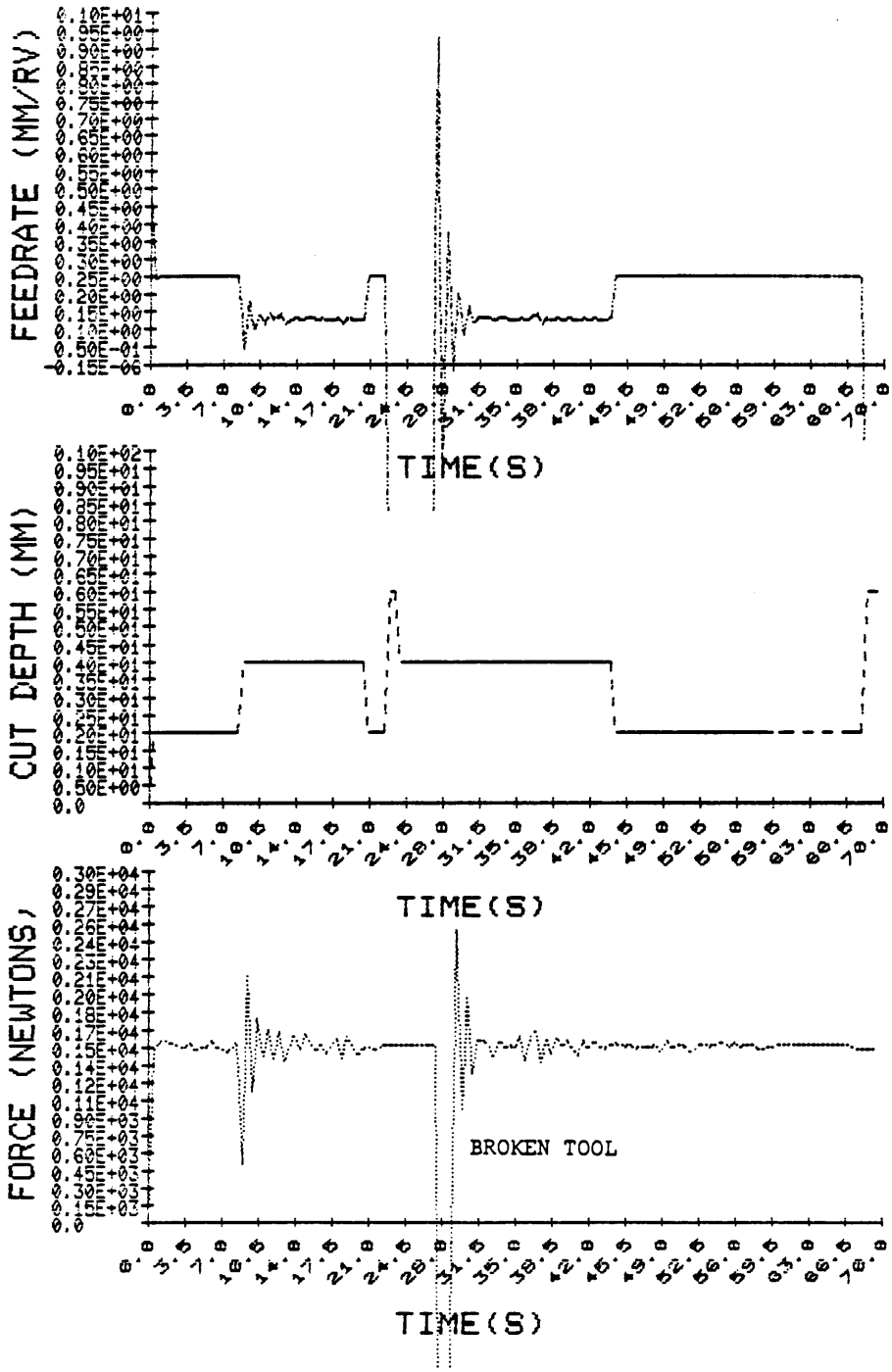

Figure B.5. Cutting Trace With Broken Tool (600 rpm)

APPENDIX C

ADAPTIVE MACHINE CONTROL USING OPTIMAL TRAJECTORY

Optimal adaptive control of machining operations is becoming more cost effective due to the increased processing speeds of microprocessors and A/D sensor processors. This is especially true in CNC machines where input and output control signals are binary, enabling the optimal control processor to be directly integrated into the system. Perhaps the greatest potential for adaptive control is in CAD systems where the part program is developed on a computer and fed directly to the machine. An optimal adaptive control system can greatly reduce the overhead required of the part programmer and CAD system, since the adaptive control system can determine optimal control parameters for a given design. Adaptive control can also continuously monitor the machining process to ensure optimal machining operations given the current conditions. In addition, catastrophic machine and tool failures can be detected by the adaptive control system in advance, so corrective machine control actions can be implemented.

Despite all the advantages of optimal adaptive control, most machining systems use feedback or constraint optimal control [27]. These systems monitor the machining process and implement a control action for a predetermined condition or input/sensor error correction. Such processes are not truly optimized to a performance index. Sensor and measurement difficulties have been the main problems in implementing dynamic optimal adaptive control routines [12, 53].

In metal removal, the machining process is very harsh and noisy for instruments and electronic transducers. When the parameter information is received by the computer, the state parameters of the system are put into a model of the system, and a search routine is employed to determine the optimal control responses according to the model's performance index. By the time the search is made, the initial machining conditions could have changed drastically and optimal control decisions would no longer be valid. In addition, variability in the tool and work material can make the model control decision erroneous, possibly driving the system into an unstable state.

An optimal adaptive control system is developed for a turning process. The control optimization involves dynamically tracking a tool wear trajectory based on an optimal tool life for the particular material and machining process. The optimal tool life is computed off-line for the performance index. The control system is designed to use the total capability of the microprocessor system architecture. Real-time numerical integration is performed, taking advantage of microprocessor register manipulation speed. The adaptive control system is simulated using the dynamic model developed in Appendix B.

System operating constraints and catastrophic failures are serviced as interrupts. These interrupts prevent the optimization routine from driving the control system to an unstable or hazardous state. This adaptive control design takes full advantage of the latest developments in microprocessor support systems. Real-time optimization routines can make optimal control decisions without using

time-consuming search techniques which require mass memory. The optimization routine is disabled immediately when a constraint is violated or an impending catastrophic machine or tool failure is detected. Most important, the control system is simple both from a hardware and software standpoint. System implementation is very cost effective, especially on a CNC machine.

Adaptive Control Systems and Optimization

Optimal Adaptive Control for Machining Processes

The basic adaptive control process is presented in Figure C.1. Sensors measure the performance of the machining operation. Measurement signals are amplified, filtered, and converted from analog signals to binary words which can be read by the processor. This system is a logical control system and can be run in parallel with a feedback control loop system. The feedback loop will keep the system stable during control changes. The microprocessor and logic hardware determine the optimal machine response as a function of the measured inputs. Once a decision is made concerning the optimal response, a control message is sent to the decoding hardware, where it is transformed into control signals for the servos and drive motors.

Offline Performance Index

A performance index must first be established to determine the configuration of the adaptive control system. Using this index, the optimal performance of the system can be determined. Many performance indexes and optimization routines have been developed for process

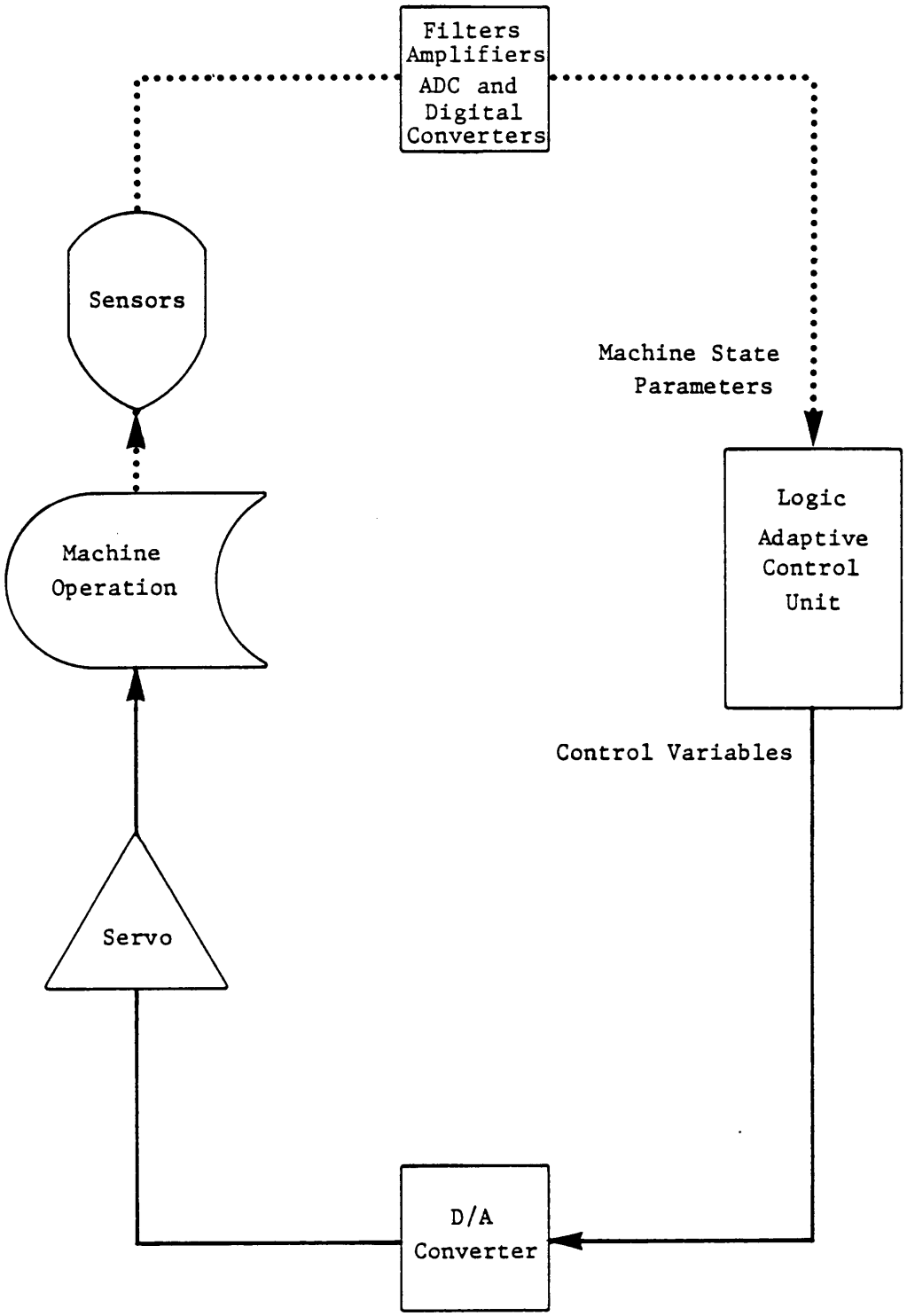

Figure C.1. Basic Adaptive Control System

optimization [13, 26, 31, 75, 81]. These routines usually use feed and speed as control variables, while depth of cut is assumed to be fixed. The routines use tool life as the major characteristic variable. These routines determine tool life and optimal control variables (feed, speed, and depth of cut) offline before the machining process begins.

Some online optimization and measurement routines have been developed [76]. Measured data from the sensors are put into a model of the machining process for the given part material and tool type. Search techniques or state values are used to determine the optimal speed and feed for measured conditions. Optimal control conditions are determined and machining control parameters are changed to enable the system to obtain the optimal state. A major problem in these routines has been with dynamic optimization; nonlinear search techniques require a great deal of processor time. Mass memory is needed and control actions can drive the system to unstable conditions.

Basic Adaptive Control Design Concepts for Optimal Trajectory Control

The adaptive control strategy being developed uses an offline optimization routine given a performance index to obtain the tool life for a given machining process and part material. The tool life defines the optimal trajectory over time for the machining process and workpiece material. The tool is replaced at the end of its projected life. The offline routine provides initial cutting speed, feed, and depth of cut. An optimization routine tracks the optimal tool life trajectory. Speed and feed are adjusted to maintain the optimal track.

The optimization process is subject to constraints (e.g., power, tool thrust, vibration) which define the feasible operating region. Constraint interrupts in the system prevent the system from operating outside constraint limits.

The constraint control strategies are presented first. The optimal control routine is discussed and, finally, the hardware requirements are presented. The system is developed for a turning process but can be applied to other machining operations.

Constraint Control Strategy

Catastrophic failure is one of the most costly types of tool failure. When the tool breaks, usually the workpiece is destroyed by the machine. Motors are burned out when power requirements are exceeded. Machine damage occurs when spindle torque is exceeded. Catastrophic tool failure cannot be predicted and occurs very suddenly. For these reasons, the adaptive control system must respond immediately to sensors, indicating that an operating constraint is or is about to be violated.

All constraint sensors in the adaptive control system are connected to the microprocessor's hardware interrupts. The constraint signal trigger level is set by the processor before the process begins. When a sensor voltage or current reaches the trigger level, the interrupt is activated. The main microprocessor immediately stops the program it is currently processing, saves the information, and goes directly to the beginning of the constraint program. The constraint program's first instruction is to reduce the critical control

parameters (feed and speed) to drop the operating system from the constraint boundary. Feed and speed are reduced simultaneously to ensure that surface finish is not degraded. The flowchart of the basic algorithm is presented in Figure C.2. The program keeps checking the sensor measurement data until the signal goes below the trigger level. A program down counter counts the number of times the signal is checked and the control variable is reduced. If the control variable is reduced to a minimum level (preprogrammed in the interrupt service routine counter), the system is shut down completely. A major failure has occurred or is about to occur, and the tool is retracted from the workpiece. If the trigger signal can be reduced, the optimization routine is reinitiated on the next sample clock pulse.

The interrupt-driven system ensures that constraint conditions are serviced on the next instruction cycle, after the program counter and flag contents are pushed on the stack. The interrupts can be prioritized so that conditions which will cause damage to the machine can interrupt other interrupt service routines not having as severe damage potential. By interrupting the optimization program, the optimal control decisions can never drive the system beyond the constraints. This is an important characteristic of the system and integrates the constraint and optimal adaptive control systems. The microprocessor is most efficiently used in this configuration.

Some of the constraints which can be measured are power with a power transducer, tool breakage with a piezoelectric vibration transducer, chip buildup with a tool thermocouple, speed with a

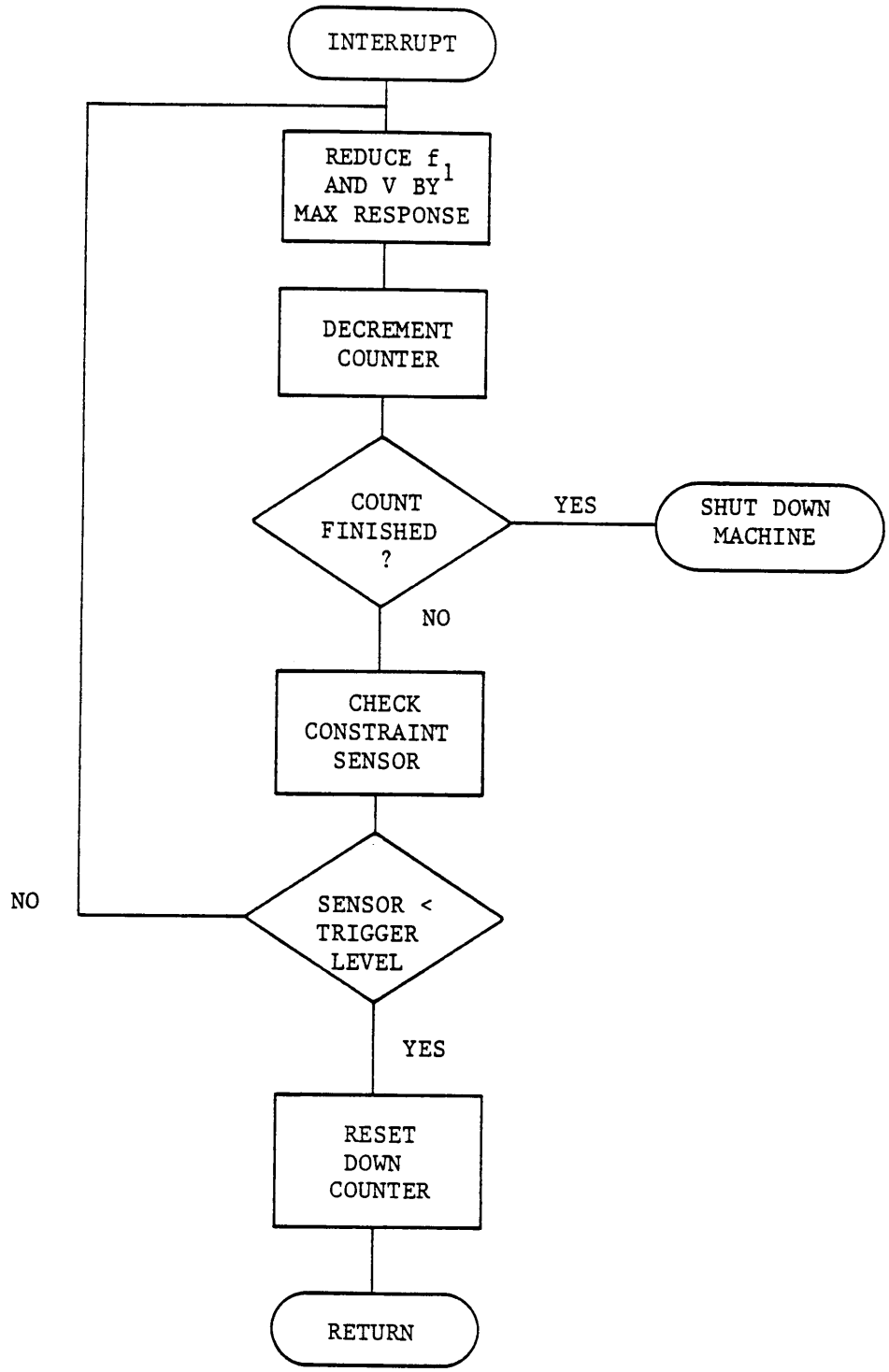

Figure C.2. Constraint Interrupt Service Routine Flowchart

tachogenerator, feed with a resolver, and spindle torque with a dynamometer. Differential control variables, such as acceleration, provide reliable information for determining impending tool failure.

Transducers can be connected to signal processing equipment which triggers a signal to the microprocessor when the signal gain reaches a certain limit. This trigger limit can be set by the processor in some equipment to allow for constant differences in the workpiece material. For instance, different cutting tools will have dissimilar vibration frequencies. So the threshold level for a piezoelectric transducer could be changed as a new tool or work material is machined. This capability adds extra flexibility in the adaptive control system and ensures that maximum cutting use can be obtained from the machine.

Optimal Control Strategy

The optimal control strategy used in the adaptive control logic unit is a trajectory optimization routine. The optimal trajectory is established offline, based on the performance index and characteristics of the system. Using measured input data at specific uniform time intervals, the actual trajectory is determined by numerical integration. A control decision is made to return the system to the optimal trajectory in the minimum amount of time. The actual trajectory is checked at the next sampling time interval. The process continues until estimated tool life is reached, and a new tool is installed on the machine.

Optimal Trajectory

Optimal tool life must first be estimated for the cutting condition and part material. Many methods have been developed to determine optimal tool life given the part material and machine. These methods are based on minimum production time, minimum cost or profit models. Most of these methods also provide an initial speed and feed rate and an optimal depth of cut. Thus, feed and speed are the parameters which are assumed to be controlled during the process.

The machining control parameters would be optimal through the entire tool life if the tool stayed sharp and there were no variation in the workpiece material. However, this case does not exist in a real machining environment. Tool wear, hard spots, tool and machine variation cause different forces and cutting conditions in the work material. Thus, feed and speed must be considered to vary with time.

The tool life can be described as:

$$W_F^* = \int_o^T (dW_F/dt)dt \qquad (1)$$

where:

T: tool life

W_F: tool wear (flank)

W_F^*: maximum tool wear

Flank wear dW_F/dt is a constant rate once initial wear W_{FI} is taken into consideration [36]. Initial flank wear occurs when the tool first enters the workpiece and has not reached steady-state temperature [71]. This initial wear is considered to take place instantaneously at time equal to zero. The initial wear is approximately 0.05 millimeter

for most types of materials [76]. The change in wear over the tool life is:

$$W_F^* - W_{FI} = dW_F/dt \int_o^T dt \tag{2}$$

$$(W_F^* - W_{FI})/T = dW_F/dt \tag{3}$$

Direct measurement of tool wear is extremely difficult. One method uses a current and voltage thermocouple configuration to determine tool wear [15]. An empirical relationship was established between tool wear change and speed, feed, and thermocouple reading. This measurement technique is very dependent on the cutting properties, signal noise, and variation from tool to tool.

Cutting force is probably the most conventional and reliable measurement which can be related directly to tool wear [76] . Using tools with groove design, change in cutting force with respect to tool wear is almost constant and is proportional to the depth of cut. The relationship is:

$$(1/d) \times (dF_c/dW_F) = K_1 \tag{4}$$

This relationship is independent of the cutting speed and feed given a specific workpiece material [74]. The workpiece constant k_1 is linearly related to the Brinell hardness or yield stress and is given by:

$$k_1 = k_2 H_B \tag{5}$$

where:

H_B = Brinell hardness

k_2 = constant dependent on the material

Changes in k_2 are independent of time:

$$(dW_F/dt) \times (dF_c/dt) = dW_F/dF_c \tag{6}$$

Combining equations (3), (4), (5), and (6), the change in cutting force F_c with respect to time is derived:

$$dF_c/dt = dk_2H_B(W_F^* - W_{FI})/T \tag{7}$$

This is the optimal trajectory of tool wear with respect to time. The slope is constant (a line between two points). From equation (7), the optimal trajectory can be determined from material hardness, depth of cut, maximum flank wear, and tool life. This becomes the optimal trajectory used by the control system to determine system performance. The force can be easily integrated (summed) over time by adding the change for the time period to the accumulated change for previous time periods.

Machining Constraints

The control strategy is to maintain the optimal trajectory within machining constraints. As the tool's cutting edge dulls, the cutting force must be increased. Cutting force is a function of feed rate and depth of cut and is expressed as:

$$F_c = k_F f^{\times} d^{\beta} \leq F_{cmax} \tag{8}$$

where:

F_{cmax} = maximum cutting force

$k_F, \times, \beta$ = constants reflecting the workpiece material

The depth of cut is constant during the cutting process. Thus, the change in cutting force with respect to feed rate is a function of a power. This nonlinearity can present a problem when linear control systems are used. The computation of numbers to the powers is one of

the longest function execution times for the microprocessor. This computation time is in the same neighborhood as the computation time for the transcendental functions (40 to 45 milliseconds). A lookup table for the log values might be required if computational speed is critical.

Another important constraint is surface finish. When the feed rate is adjusted to compensate for tool wear, the control action must consider the following surface roughness constraint:

$$f^2/8R \leq R_{max} \tag{9}$$

where:

R - nose radius of the tool (mm)

R_{max} - maximum roughness

In addition, speed must be kept in a specific range with respect to feed rate. The normal operating range can be expressed as:

$$fV^{\gamma} \geq S \tag{10}$$

where:

γ and S are constants

($\gamma \simeq 2.0$ and $S = 1200 \sim 1400$)

By operating in this range, edge buildup is minimized.

The final constraint to consider is power. This constraint is:

$$P_c = (k_F f^{x} d^{\beta} V)/(6120\eta) \leq P_{cmax} \tag{11}$$

where:

η = machine efficiency

P_c = power

P_{cmax} = maximum power output

With all these constraints, plus the maximum and minimum feed rates, a convex set of boundary conditions is imposed on the machining operation. By using logarithmic transforms, the linear constraints are presented in Figure C.3. Thus, the optimization routine must ensure that the system remains within the operating constraints after the control action reaches steady state. For example, if feed rate is increased along the power constraint boundary, velocity must be reduced. The optimization routines which determine initial feed and speed are usually constrained by surface roughness and power, because the maximum metal removal rate occurs in the region. Metal removal rate is proportional to feed and speed.

Control Strategy

Once the optimal trajectory and constraints have been established, the optimal force trajectory can be tracked, based on the current position of the system with respect to time. The optimal trajectory is determined by the microprocessor. (The actual tracking of the trajectory at equal time intervals by the microprocessor is discussed in the following section.) Thus, at any time during the cut, the optimal force trajectory is known. The actual force is measured from a tool head strain gage at equal time periods. In addition, the force change from the preceding time period is determined by subtracting the present cutting force measurement from the measurement in the preceding sample time period. This gives the "instantaneous" change over the time period. If the cutting force is within the limits of the estimated trajectory, the feed is not changed. The speed is checked to

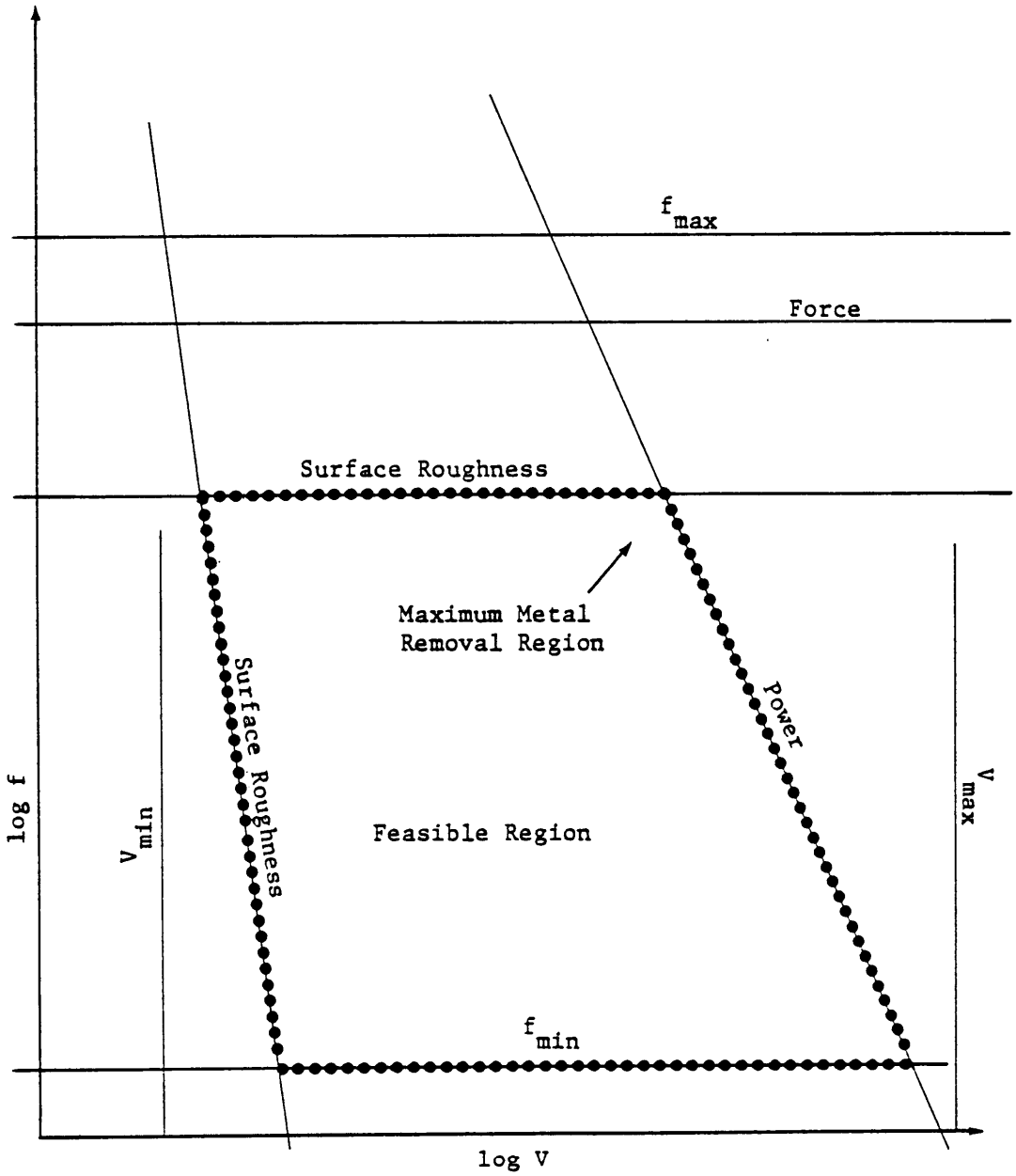

Figure C.3. Boundary Constraints

ensure it satisfies the surface roughness constraint expressed in equation (10). Feed is adjusted to bring the system into compliance with the roughness constraint.

From equation (8), feed is the only factor that can be directly related to the cutting force trajectory. Thus, feed is considered first by the optimization routine. The routine (Figure C.4) first subtracts the optimal trajectory force from the current trajectory force. If the result is within the trajectory limit, then the speed is checked for control actions, and the cutting force is not changed. Otherwise, the slopes of the two trajectories are compared to check for a possible intercept. If there is an intercept with the optimal trajectory, the feed rate is changed to drive the current trajectory toward the optimal trajectory. Otherwise, the feed rate correction must be determined. It is possible that a control feed rate action could cause the system to overshoot the optimal trajectory.

The result of a control action is presented in Figure C.5. It is assumed that feed rate and speed can be changed by an incremental amount $\pm f_p$ and $\pm V_p$ during the optimization period. The amount of time the system takes to reach these steady-state values is τ_f and τ_V. Thus, a control or no control decision is made by the routine. This type of adaptive control system is especially compatible with reference-pulse and sampled-data CNC machines. The adaptive control system can take advantage of the basic length unit (BLU) and timing pulses on CNC machines.

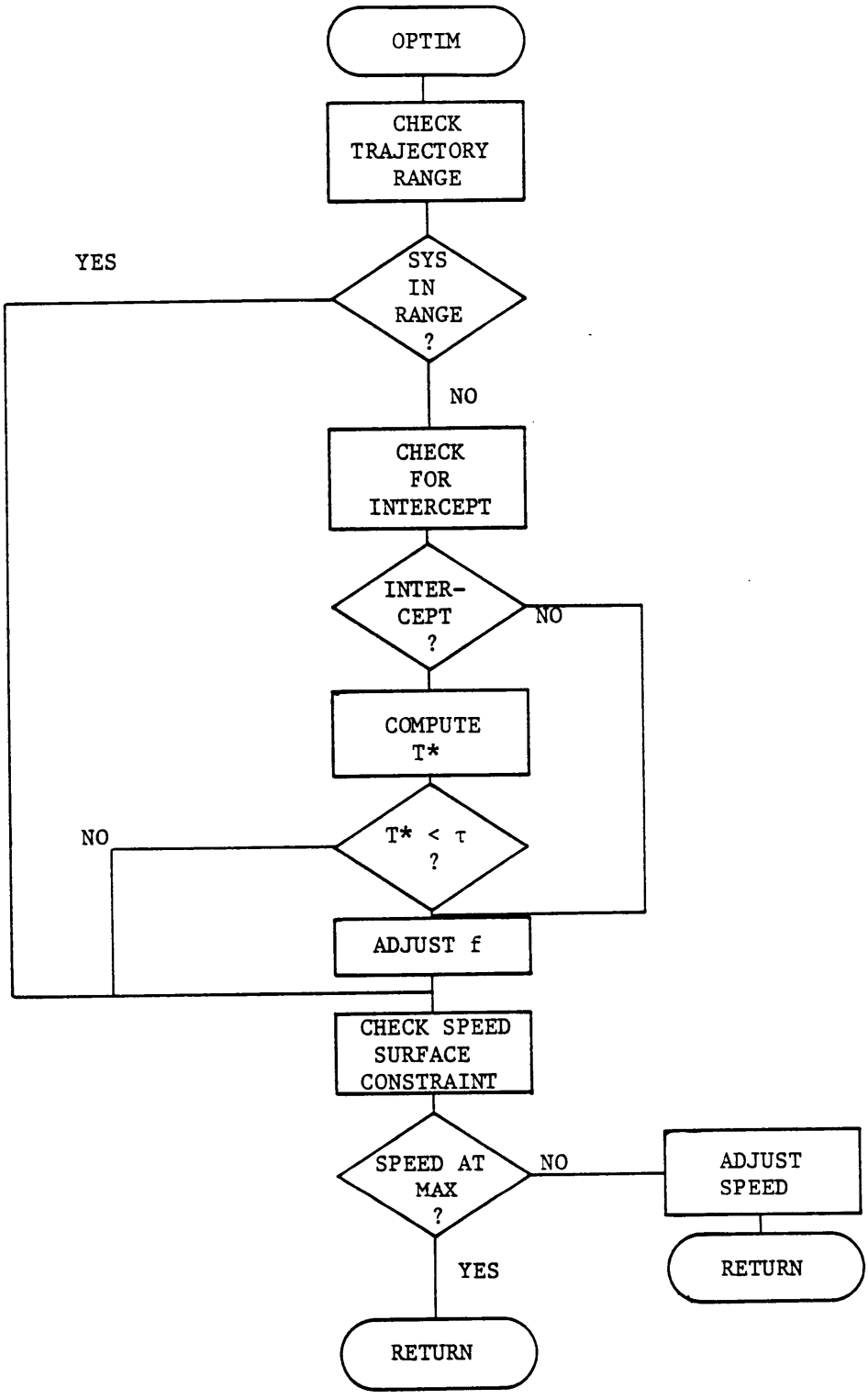

Figure C.4. Optimization Routine Flowchart

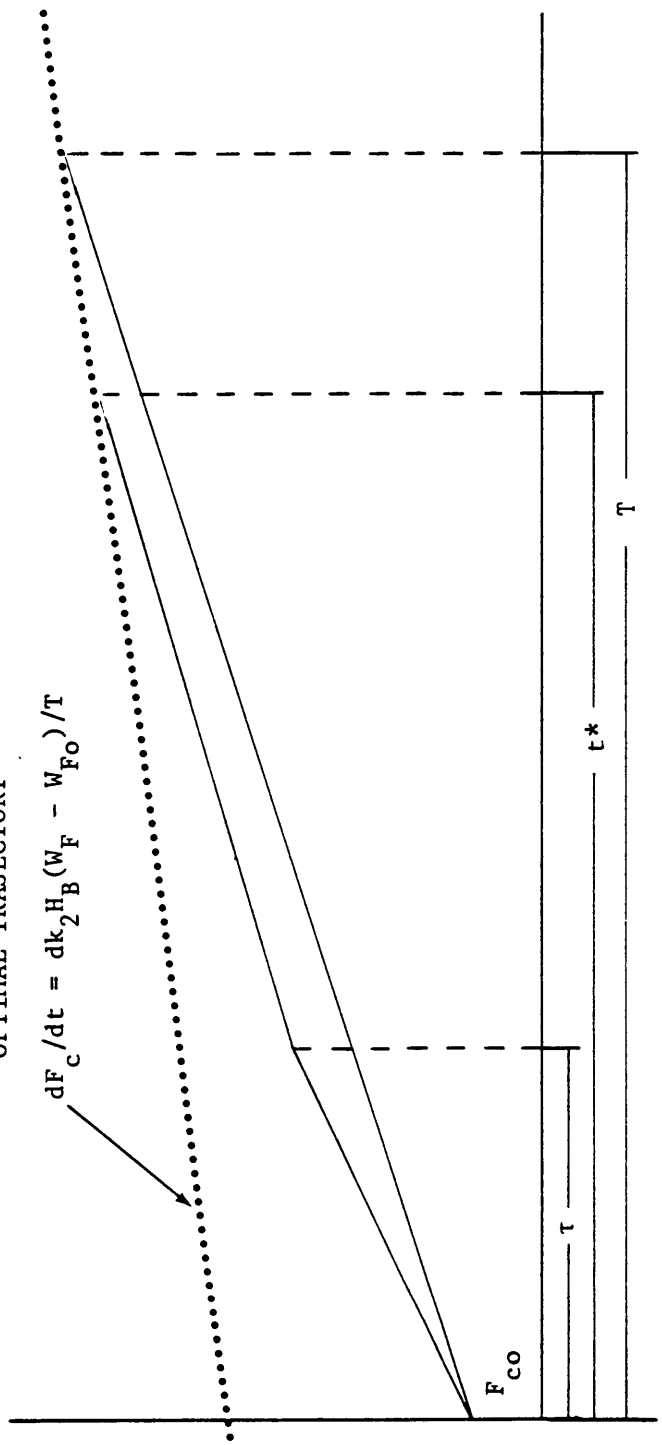

Figure C.5. Control Action Trajectory

As Figure C.5 shows, a control action was implemented to reach the optimal trajectory in the minimum amount of time. The optimal trajectory force F_c^* at time t_i, the measured force F_{co}, and slope of the current trajectory F_{co}/dt are known. The time T for the system to reach the optimal trajectory is determined. (The feasibility of intersecting the optimal trajectory has already been determined.) Using the control action, an intercept is computed by the following relationship:

$$t^* = (F - F_c^*)/\{(dF_c^*/dt) - (dF_{co}/dt)\} \qquad (12)$$

If the minimum time t^* with control is less than or equal to the normal intercept time T, the control is implemented (provided surface finish constraints can be satisfied).

The speed is always increased if the surface finish constraints are satisfied as described in equation (10). The τ_f and τ_V are set equal to each other for simplifying control. If the speed cannot be changed, the feed rate control action will not be implemented when the surface finish constraint is violated. Note that power is an interrupt constraint and automatically lowers the speed when violated. The system continues the optimal control during equal intervals until the part is finished or the optimal tool life is reached.

System Configuration and Operations

Configuration

The hardware used in the adaptive control system takes advantage of the latest microprocessor system technology. The system uses a 16-bit high-speed microprocessor operating around 8 MHz. In addition, an input/output processor (IOP) is employed to handle most of the I/O processing load and direct memory access (DMA). An interrupt controller is used to handle all sensor interrupts and prioritize them. The priorities can be changed by the microprocessor to reflect new cutting materials and machine configurations. A sample system configuration is presented in Figure C.6. Intel 8086 microprocessor system components are indicated in the illustration [49]. The system can be developed with other microprocessor systems.

Operation

The optimal control program is stored either in random-access memory (RAM) or read-only memory (ROM). The optimization program is activated by a down counter clock. The time to count down to zero is equal to the interval between data samples. A "one shot" is sent to the IOP, which immediately obtains the sensor data from the A/D converters. Due to slow conversion rates of sensor signals, a long delay can occur before the signal is ready for transmission to the processor. The estimated time is approximately 150 microseconds for a high-speed system.

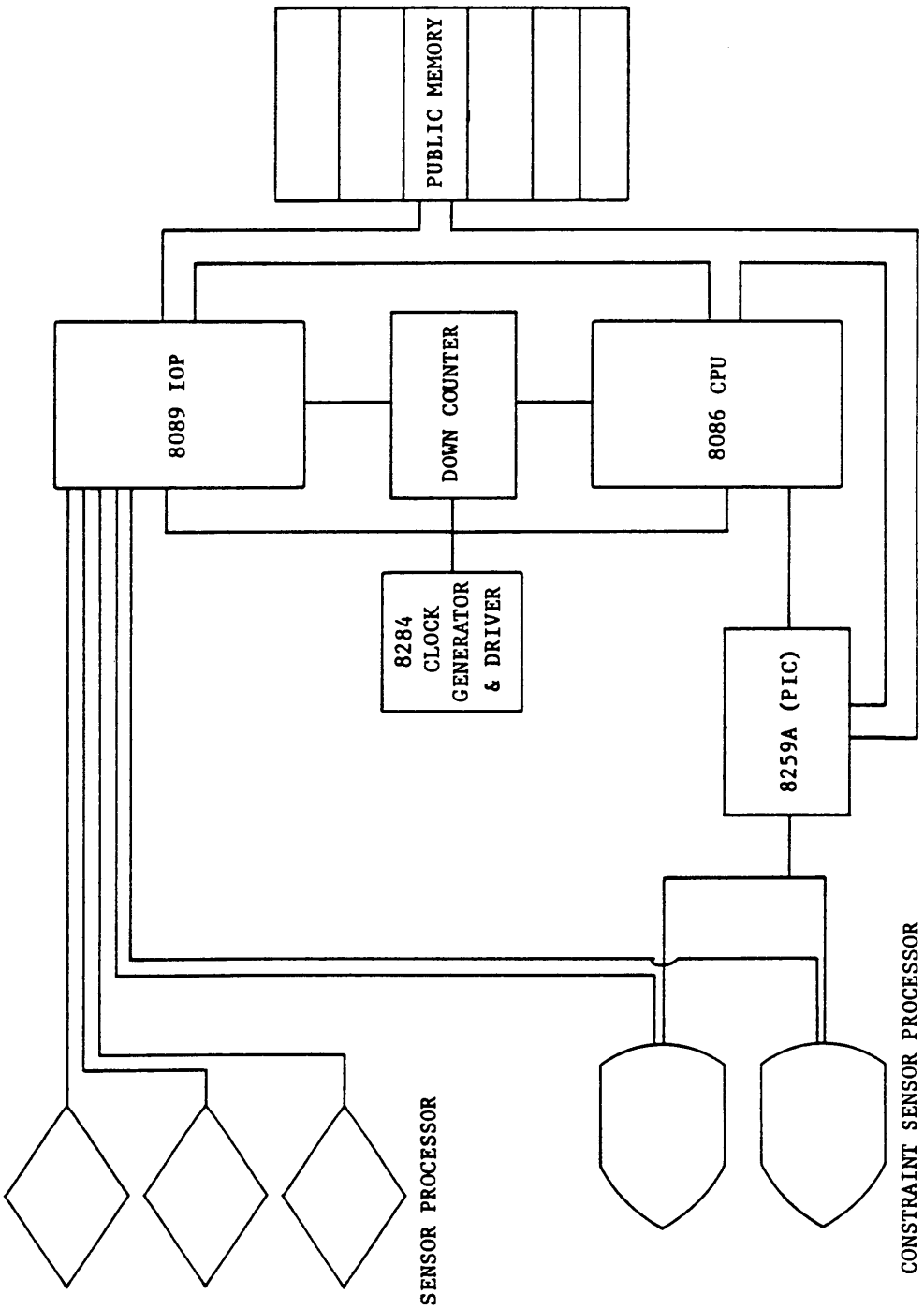

Figure C.6. Hardware Configuration

While the controller program is waiting for A/D conversions, the program in the controller adds an increment representing the change in the optimal trajectory to an appropriate memory location. This is equivalent to numerical integration.

Once sensor data has been transferred to memory, the processor begins the optimal trajectory program. Most of the data manipulation is performed in the registers. Thus, only 50 microseconds are required to perform the optimization routine and send the control message to the servodrives.

To obtain a perspective on the measurement and computation for the optimization routine, consider the following worst-case machining process. A turning operation working at 600 rpm rotates approximately 0.01 revolution in 500 microseconds. If the feed rate is 0.75 millimeter per revolution, the tool would have moved a distance of 0.0075 millimeter during this time period. The system response time (expressed in milliseconds) is extremely slow compared to the decision process. Thus, the system might make several revolutions before the control actions reach steady state. A decision must be made according to the machine response and variability in the machining process. This system can respond in a minimal amount of time compared to most machining operations.

The response time to interrupt constraints is approximately 120 microseconds. Thus, if a hard spot is encountered, the control servos would receive commands before the tool could move more than 0.001 millimeter once the hard spot is detected. The sensor response

time to detect hard spots is usually much greater than this computation time [83]. This is especially true with thermocouples where temperature response times are delayed due to heat capacitance and diffusion of the tool. However, the constraint interrupt response routine can provide a very fast response time with respect to machine response time.

Real-Time Operation Example

The following example illustrates the entire adaptive control system execution from startup through several control sampling periods.

A mild steel workpiece is being used in a turning operation with a carbide tool. The following operational data is used to determine an optimal feed, speed, depth of cut, and tool life with an offline procedure [76]:

$x = 0.72$, $\beta = 0.75$, $P_{cmax} = 2.25$ kW

$f_{min} = 0.3$ mm/rev, $f_{max} = 0.75$ mm/rev

$V_{min} = 5$ m/min, $V_{max} = 400$ m/min

$F_{cmax} = 60N$, $K_F = 115$

HB = 150, $k_2 = 0.10$, $W_F^* = 0.3$, $W_{FO} = 0.05$

Initial optimal control states are:

$d^* = 2.5$, $F^* = 0.75$, $V^* = 114.8$

$T = 25$ min, $F_{co} = 194$ kg

Note the initial cutting force ($F_{co} = 194$ kg) was calculated using the initial feed and depth of cut in equation (8). This value can also be obtained from empirical data. Still computing offline, the slope of the optimal trajectory is determined from equation (7):

$$dF/dt = 0.0083 \text{ g/msec}$$

Thus, the trajectory is computed offline. From machine data, the change in feed and speed is 0.015 millimeter and 0.07 meter per second, respectively, with $\tau_f = \tau_V = 0.1$. The sampling interval was selected to be τ_f time or 100 milliseconds. Note that the machine servo and motor responses have the slowest response characteristics. Some CNC machines have $\tau_f \approx$ 120 milliseconds [35]. From equation (8), the cutting force can be changed 11.6 grams in 100 milliseconds. Different cutting force rates can be assumed directly proportional to the depth of cut [74].

The change in the optimal trajectory per sampling period is 0.83 gram per 100 milliseconds. This information is stored with the response data in the microprocessor memory. The offline computation has to be made only once. The computer optimization program is also stored in memory. Because of the small memory requirements and small program size, over 1500 machining and part material data sets can be stored with the optimization and constraint programs on 8 kilobytes of 8-bit memory.

An example control procedure is presented in Figure C.7. Between points A and C, a normal optimization control sequence is encountered with an increase in feed (force) applied at point B. The optimization routine is interrupted by a power interrupt. Power and feed are reduced. When the next optimization parameters are sampled, the actual

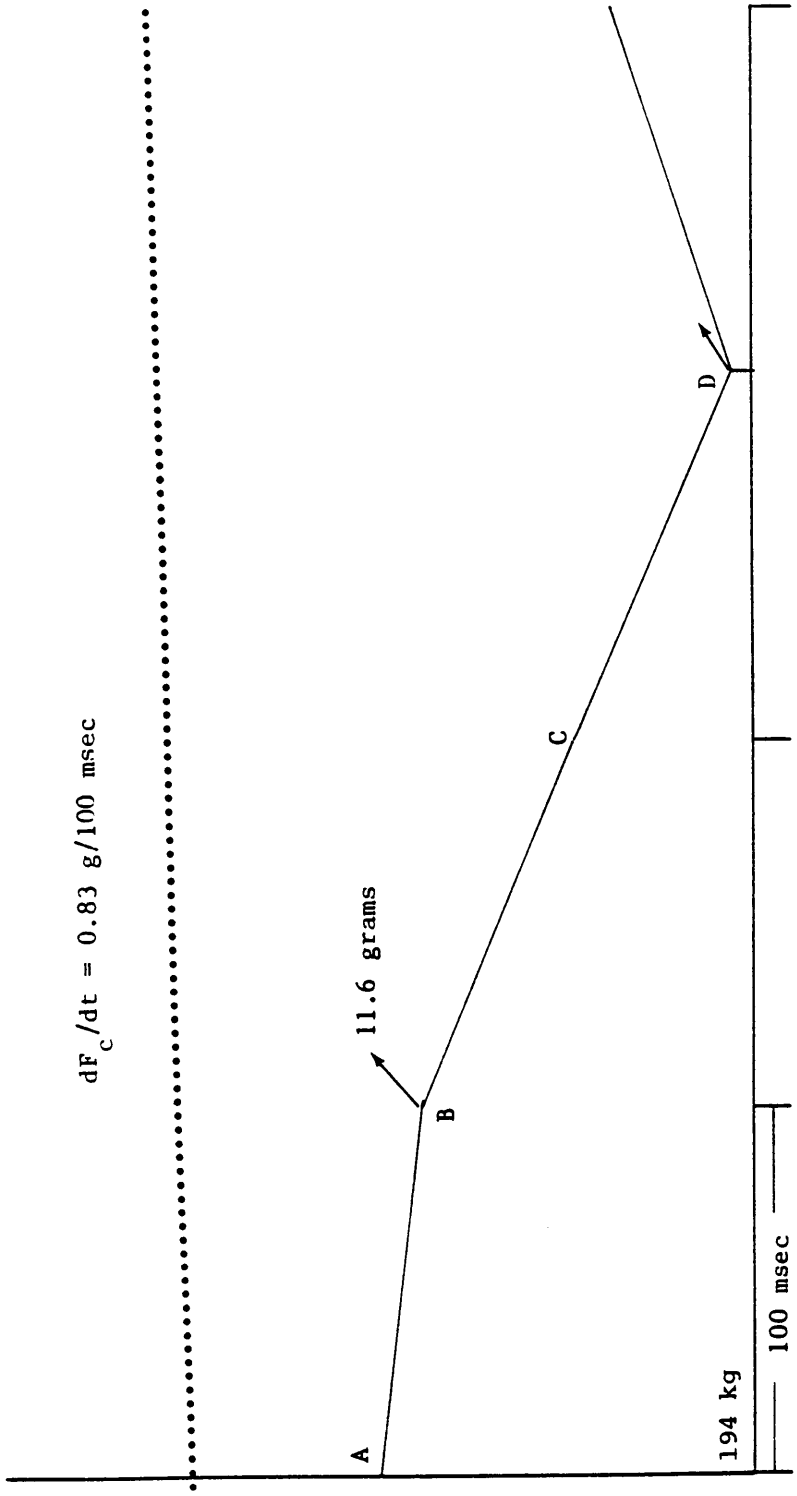

Figure C.7. Adaptive Control Action

force trajectory has dropped drastically. The actual force is being measured at each sample interval. The rate of change in force is immediately increased by the optimization program. The process continues until the part is completed or tool life decreased.

Summary

The adaptive control system strategies and hardware have been developed to enable fast response to the machining process. New and advanced microprocessor technologies (such as DMA and IOP) enable quick access of sensor information. The interrupt constraint system can maintain the system in the constraint boundaries. These constraint boundaries can be changed as a new material is introduced or a different machining operation is performed. The register manipulation of data reduces computation time to obtain an optimal decision. Tool wear is measured directly as cutting force. The optimal trajectory is "integrated" over time by simple counters. Force measurements provide the actual position of the system with respect to time. Most of all, the system is capable of connecting a large range of sensors to detect catastrophic failures. This system design truly integrates constraint and optimal adaptive control.

System implementation is not difficult because of the simple design. Software development requirements are small, since all the optimization routines can be performed directly by the ALU. All interrupt routines are simple programs comparing one measurement device with a predetermined constraint level. The constraint routines remove most of the overhead from the optimization routine because

microprocessor capabilities are used effectively. All optimization is done offline where it can be computed more efficiently.

Flexibility is the key to this adaptive control system. As new materials are machined, new parameters and constraints can be added to the program. A controller will automatically adjust the constraint boundaries and operating parameters to reflect the new material or tool. The tool history can be recorded. This will enable a tool to be changed before the tool life is complete. Once the tool is used again, the tool's current point on the trajectory is recalled from memory and control conditions. The optimal trajectory can be changed to reflect the initial tool wear.

The rapid response time of this adaptive control system makes the servo and motor response characteristics the slowest times in the adaptive control system. If the microprocessor outperforms the machining system, a less expensive 8-bit processor could be used, thereby reducing controller cost. Machine servo and motor drive response times must be decreased for these adaptive control systems.

Adaptive Control Simulation

The adaptive control system is integrated into the CNC lathe simulation model described in Appendix B. A slight modification of the control program was required. Otherwise, the CNC system remained intact. This ease of modification illustrates the practicality of the adaptive control system.

The force summation program is changed to increment the cutting force by the flank wear factor for the prescribed optimal control track. Thus, this factor is constantly added to the reference cutting force, based on the speed and feed of the system. Reference cutting force variations are directly compensated, since feed and speed vary with the force and cutting material conditions. Feed and power limitations are automatically checked by the CNC internal control system.

The adaptive control system is simulated using the CNC lathe and material characteristics described in Appendix B. Flank wear was based on the force and depth of cut. Figure C.8 is the output of the normal force control routine with actual-to-flank wear simulated. Machining time is 228 seconds. Flank wear increased cutting force. The control system decreases the feed rate to compensate for this increase in force. The feed rate plots indicate the decrease in speed.

The adaptive control system was added to the simulation model, using the same material and cutting conditions as in the force control simulation. Simulation results are shown in Figure C.9. As the phase shift indicates, cutting time is reduced by 68 seconds with the adaptive control system. As illustrated by the example, this adaptive control system has great potential for optimal control of machining operations, based on the total machine cycle performance index. The compensatory tracking system can be applied to other machining processes, as well as robotics. The simplicity of the control system makes this a practical digital control application.

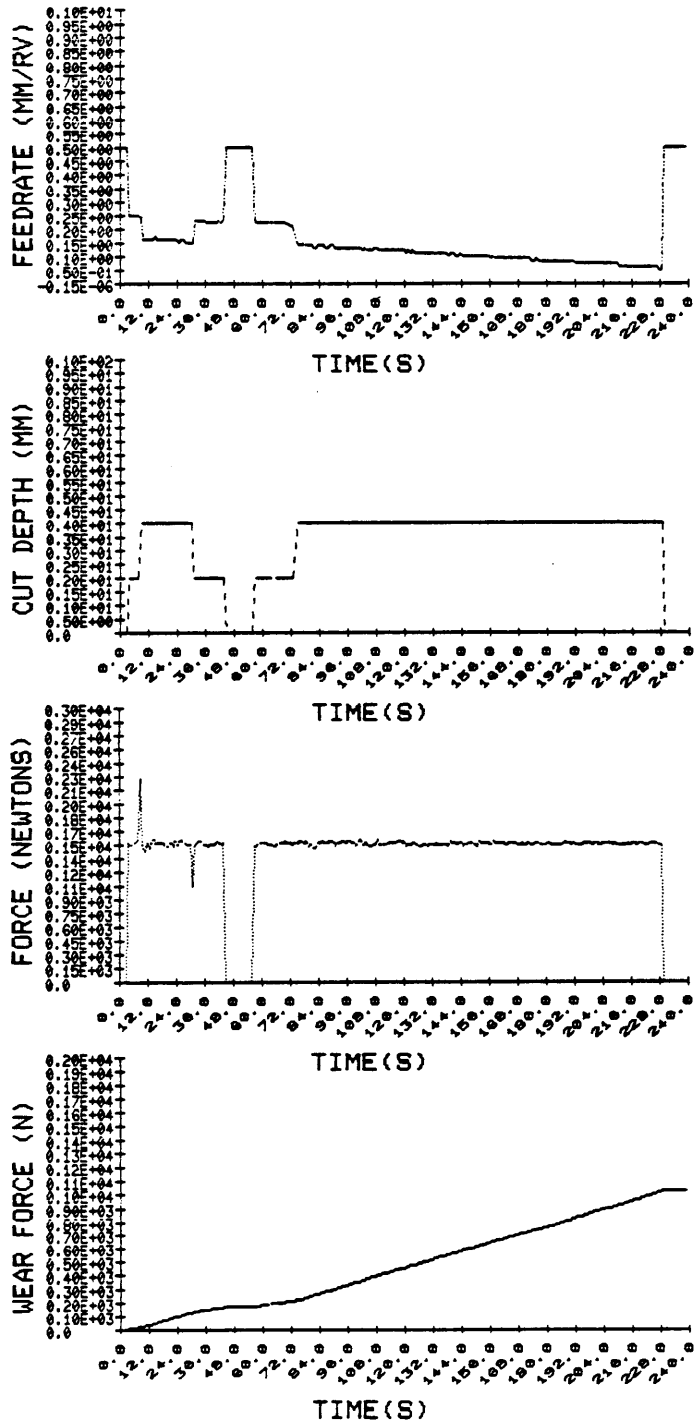

Figure C.8. Force Control System with Flank Wear

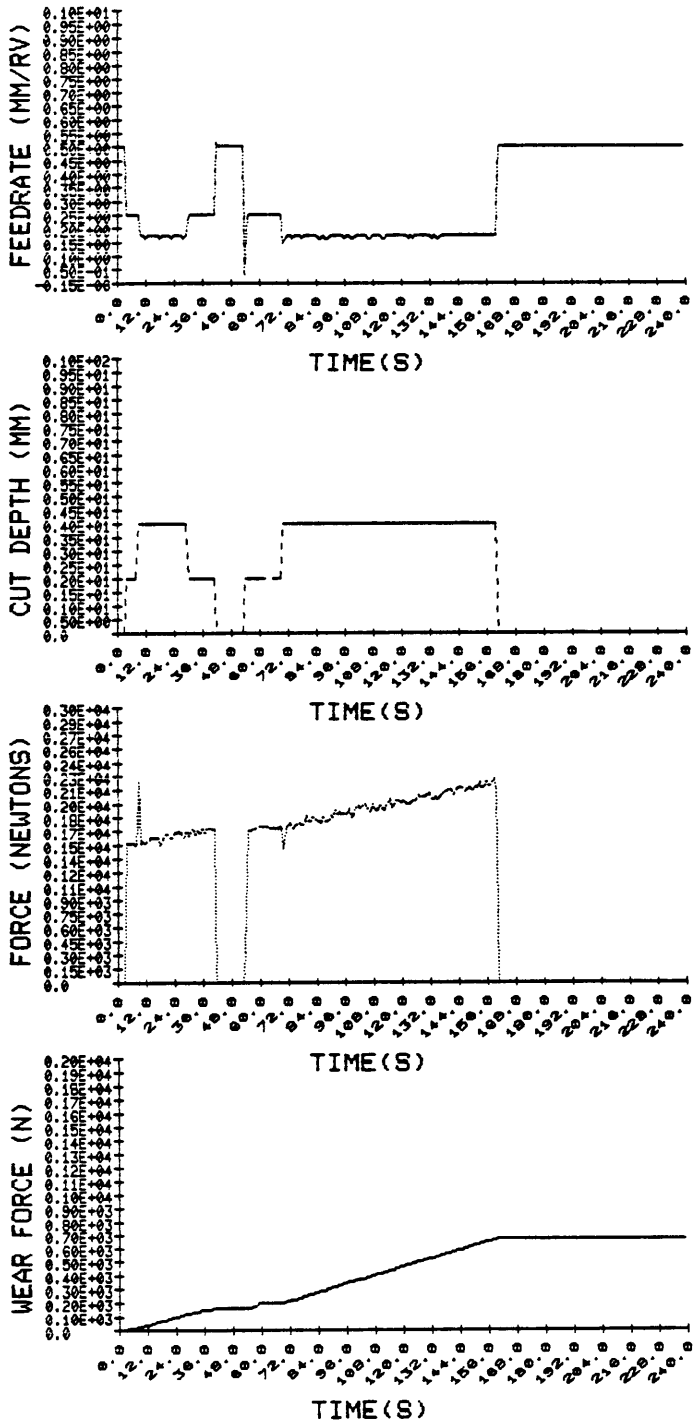

Figure C.9. Adaptive Control System with Flank Wear

The vita has been removed from the scanned document

DESIGN AND EVALUATION METHODOLOGY FOR

COMPUTER-CONTROLLED MANUFACTURING SYSTEMS

by

Harold A. Scott, Jr.

(ABSTRACT)

A methodology is developed to determine cost-effective hierarchical computer control network designs for flexible manufacturing systems. By modeling the hierarchical control system (HCS) as a resource allocation problem, an optimal hardware configuration is identified using dynamic programming. Being independent of specific computer hardware technology, the model can address present and future automated manufacturing systems.

A simulation model is developed to evaluate operational dynamics of the specified system configuration, analyze HCS component performance characteristics, and evaluate hardware and software in real simulation time. The model also simulates continuous system dynamics, as found in optimal adaptive control systems.