AN ABSTRACT OF THE THESIS OF

Chi-Shih Wu for the degree of Master of Science in Industrial Engineering presented on February 7, 1996. Title: Design and Implementation of a Flexible Manufacturing Cell with Real-Time Statistical Process Control Capabilities

Abstract approved: Redacted for Privacy

Sabah U. Randhawa

A modern Flexible Manufacturing Cell (FMC) usually consists of several Distributed Control Systems (DCS). Modeling and evaluating a manufacturing integrated system and then translating the models to operational programs are important steps in implementing a FMC. Monitoring and enhancing a FMC's manufacturing process quality is another critical issue. An embedded real-time Statistical Process Control (SPC) system can substantially reduce the delay between the occurrence and detection of an out-of-control condition during manufacturing, thus resulting in consistently high quality products.

This research develops a modeling framework for integrating the operation of FMCs with real-time SPC capabilities. This is accomplished in four phases: (1) specifying FMC's physical components, and process and functional requirements, (2) designing FMC's integrated manufacturing process and control system for individual DCSs, (3) evaluating the system design, and (4) implementing the system design. Tools used in the modeling and evaluation phases include the Scenario Integration Tables and Petri nets. A flexible communication interface for control computer and associated DCSs has been designed to implement this system framework. The $\overline{X}$ and R charts are used for real-time SPC operation of the FMC. An efficient algorithm is designed for the real-time SPC check using a finite state representation; this algorithm is then implemented using an object-oriented model.

Design and Implementation of a Flexible Manufacturing Cell with Real-Time Statistical Process Control Capabilities

by

Chi-Shih Wu

A THESIS

submitted to

Oregon State University

in partial fulfillment of

the requirements for the

degree of

Master of Science

Completed February 7, 1996

Commencement June 1996

Master of Science thesis of Chi-Shih Wu presented on February 7, 1996

APPROVED:

Redacted for Privacy

Major Professor, representing Industrial and Manufacturing Engineering

Redacted for Privacy

Chair of Department of Industrial and Manufacturing Engineering

Redacted for Privacy

Dean of Graduate School

I understand that my thesis will become part of permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Redacted for Privacy

Chi-Shih Wu , Author

ACKNOWLEDGMENTS

First of all, I would like to express my gratitude to my major professor Dr. Sabah Randhawa. Without his guidance and encouragement, I would not have reached my goal by now. I also greatly appreciate Dr. Sheikh Burhanuddin's instruction. He brought me to the world of CIM and CAM with generosity and kindness. They both not only advised me in my study, but also gave me their warm hearted concern and friendship. They served not only as my academic advisors but also as my mentors. I would also like to express my thanks to Dr. Ken Funk for being on my committee and for his suggestion for my thesis.

In addition, I would also like to thank my friends. Yu-An Li helped me in the Puma robot and FMC hardware set up, Zhong-Kai (John) Xu helped me with statistics and SPC, and Carlos Capps' shared his industrial experience in FMC with me.

Finally, I would particularly like to thank my lovely wife, Jung-Tzu and my family. Without their support, encouragement, and understanding, I would not be here to pursue my second master degree. I am thankful that this research is now complete and I am even thankful for the obstacles that I encountered during my research and my studies at OSU. They helped me grow and develop. For it is from these struggles that I learned a great deal about computer integration manufacturing and realized how much I have yet to learn.

TABLE OF CONTENTS

TABLE OF CONTENTS (CONTINUED)

TABLE OF CONTENTS (CONTINUED)

LIST OF FIGURES

LIST OF FIGURES (CONTINUED)

LIST OF TABLES

LIST OF APPENDICES

LIST OF APPENDIX FIGURES

LIST OF APPENDIX FIGURES (CONTINUED)

LIST OF APPENDIX TABLES

Design and Implementation of a Flexible Manufacturing Cell with Real-Time Statistical Process Control Capabilities

CHAPTER 1 INTRODUCTION

1-1 Motivation and Problem Statement

The concept of Flexible Manufacturing Cell (FMC) has not received as much attention as Computer Integrated Manufacturing (CIM) and Flexible Manufacturing Systems (FMS). A CIM plan is generally designed top down, whereas its implementation is generally bottom up. However, only large companies are able to implement CIM or FMS because these require large investment. Small companies are not able to offer full scale CIM or FMS operation because of their limited capital resources. Generally, small to medium companies start with FMCs with scaled transition towards FMS or CIM.

A FMC is defined as "grouping of people and processes into a specific area dedicated to the production of a family of parts or products" [Martin, 1989]. A flexible manufacturing cell is a work cell that makes use of high degree of automation and integration. This research develops an integration architecture that integrates several Distributed Control Systems (DCS) into a FMC. Manufacturing processes are described using an integrated manufacturing process scenario. This scenario is then modeled using Petri nets. High level source code for execution of manufacturing processes are then developed based on Petri net models. Petri nets play an important role in system specification and design phases. Petri nets are not only used for specifications, but also for the evaluation of the behavior of the modeled system. To implement the manufacturing process model, a communication system based on connection-oriented services is used. This is designed by International Standards Organization (ISO)/Open Systems Interconnection (OSI)'s layers 1, 2 and 7. The communication interface hardware is designed to transmit signals and is easy to expand and maintain. The communication software is able to supply users with high level function calls, which make the transition from Petri nets model to execution code easy.

Another important accomplishment of this research is the application of a real-time Statistical Process Control (SPC) within the FMC system with a Graphics User Interface (GUI) in MS-Windows environment. Most of the SPC systems for cellular manufacturing are based on off-line sampling, data collection and charting, followed by analysis of results and identifying appropriate actions for out-of-control conditions. With off-line analysis there is some delay between the occurrence and detection of an out-of-control condition.

SPC is a never-ending effort to improve a process [DeVour et al, 1991]. A process control model for quality is shown in Figure 1-1. Taguchi uses a loss function concept to quantify quality as "loss due to functional variation" as explained in Figure 1-2 [DeVour et al, 1991] . Paula [1995] indicates that "The closer you bring the monitored parameters to the root cause of the quality problem, the more money you will save through less rework or customer dissatisfaction". According to these concepts, a real-time SPC system should help operators since they need less time to do SPC tasks such as collecting data and designing and maintaining control charts. The real-time SPC will decrease the lost cost for the process because it can speed up the response time to discover out-of-control processes and reduce variations from the target. Operators can also spend more time in making parts and making process improvement. Also, a real-time SPC system can provide a track record of what really happened and its time of occurrence.

A real-time SPC system should not be limited to data collection , analysis and storage; it should offer capabilities to integrate the manufacturing hardware. Ideally, a FMC should use a real-time SPC system to do process monitoring, evaluation, diagnosis and feedback control. This can only be accomplished if the real-time SPC system is embedded into the manufacturing cell.

This research attempts to integrate real-time SPC capabilities within a FMC operation. The real-time SPC is emphasized along with the system model and its interface design for the FMC. A FMC

real-time SPC has been designed and implemented on a manufacturing cell in the Department of Industrial Manufacturing Engineering (IME) at Oregon State University (OSU).

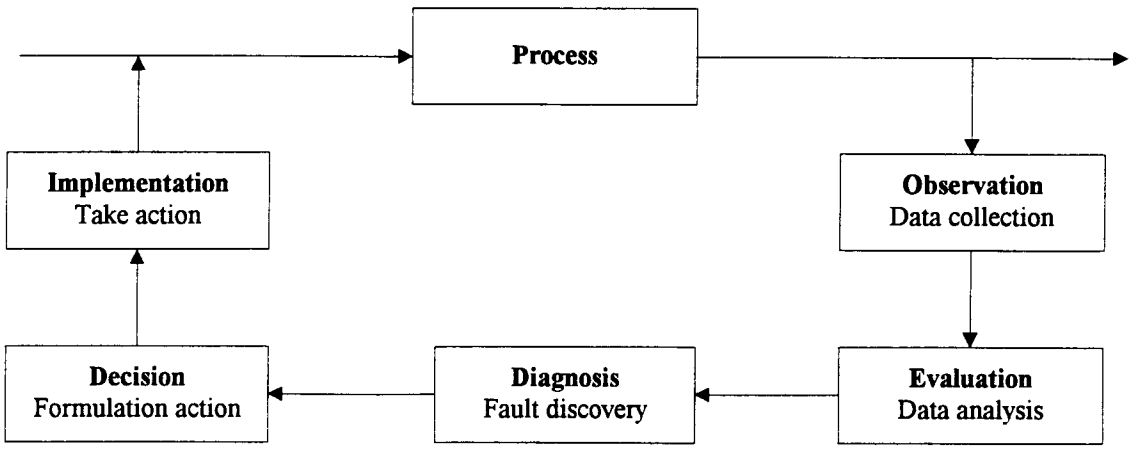


Figure 1-1 Process Control Model for Quality Control

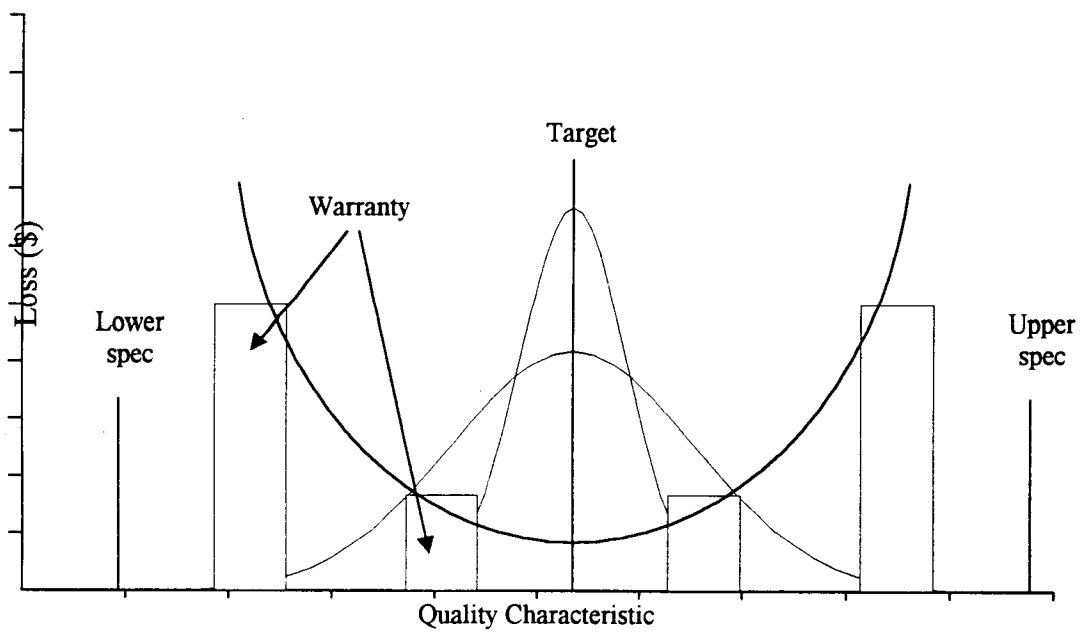


Figure 1-2 Loss Function Interpretation of the Results of the Transmission Study [DeVour et al, 1991]

1-2 Research Objectives

The objective of this research is to design and implement a computer integrated FMC with real-time statistical process control capabilities. This research focuses on strategies for designing a real-time system which can integrate several Distributed Control Systems (DCS) such as conveyors, robots and machining centers with real-time SPC functions. An $\overline{X}$ chart and an R chart are used to evaluate the process capability by using Visual C++ in a Microsoft windows environment.

The scope of this research includes the following:

- **System framework:** The generalized system framework consists of four phases. The first phase involves both requirement and specification analysis. The FMC's requirements must be specified concisely and unambiguously before starting the design process. The Scenario Integration Table and Data Flow Diagrams (DFD) are for requirement analysis and preliminary FMC design. The second phase is system design and modeling. The Scenario Integration Table is again used to design a procedure for the integrated manufacturing process. This is then translated into Petri net models. Petri nets are used to model FMC's system integration and concurrent execution. The third phase is system evaluation. Petri nets are used in this phase to analyze the manufacturing process performance and utilization. The fourth phase is implementation. This final phase involves translating the Petri net models to high level source codes which are defined function calls in a control computer system and individual DCSs. Object-oriented analysis and design are also used in designing objects for the real-time SPC software application. Flowcharts and traditional structured design skills are also used in system design.
- **Real-time SPC system design and implementation:** The FMC real-time SPC system will automatically perform data collection, data analysis, diagnosis, and decision making and its implementation. For example, when a linear trend occurs in the $\overline{X}$ chart, the system will force the machining center to replace a worn tool. For other out-of-control signals, the system will either stop

the FMC's operations for an appropriate action or will continue after displaying warning messages on the PC screen.

- **Flexible interfacing design for integrating the system:** Low-level interface controls have been designed for the input and output interface of a control computer. From these control objects, derived objects are obtained to construct an I/O bus that makes the FMC real-time SPC system flexible and economical. Also, derived objects are used to connect to a Machining Center. Boolean Algebra is employed to design mechanism for getting inputs and sending outputs from the PC's I/O addresses. RS232 serial communication in the MS windows environment is accessed to send probing information and commands from the MC to the PC.

The required design for communication using "acknowledge" makes sure the integration system is more reliable and efficient. The communication bus design, which applies ISO/OSI layers 1, 2 and 7, will allow the integration system to be upgraded with ease when the fieldbus or other advance communication systems become available.

1-3 Summary of Remaining Chapters

Chapter 2 provides review of current literature and general background information on FMCs, Distributed Control Systems (DCS) and their communication, Petri nets for modeling manufacturing cells, and SPC and real-time SPC in manufacturing.

Chapter 3 describes the integration architecture methodology developed in this research. It explains the integration framework model consisting of requirement analysis, system design, system evaluation and implementation, and provides an introduction to the tools used in developing this framework.

Chapters 4 and 5 describe the implementation of the integration architecture in Chapter 3. Chapter 4 includes a description of 1) the manufacturing system is used in this research, 2) the integration framework model for this FMC including the use of Scenario Integration Table, Petri net models and the high level operational programs for manufacturing process implementation, 3) database design and system control architecture description using data flow diagrams, and 4) interfacing and message passing methods.

Chapter 5 describes real-time SPC's design and its implementation, including the design of probing information and control chart displays and real-time statistical process control detection and handling , and implementation to the physical system described in Chapter 4. Chapter 5 also describes integration and real-time SPC tests used for system validation.

Chapter 6 provides a summary of the work accomplished and presents conclusions and contributions from this research with recommendations for future work.

CHAPTER 2 BACKGROUND

2-1 Overview

There is limited published research directly related to integrated Flexible Manufacturing Cell (FMC) with real-time Statistical Process Control (SPC) capabilities. Based on the issues addressed in this research, this review is classified into: 1) flexible manufacturing cells and cellular manufacturing design, 2) communication, integration and system testing in distributed control systems, 3) SPC and real-time SPC in manufacturing, and 4) Petri net modeling.

2-2 Flexible Manufacturing Cell (FMC)

A FMC can help smaller companies move towards the FMS level by "gradual integration" [Joseph, 1988]. Although there are many benefits associated with a FMS, such as higher machine utilization, reduced work-in-process, lower manufacturing lead times, greater flexibility in production scheduling, and higher labor productivity [Groover, 1987], FMSs are generally expensive requiring an investment in millions of dollars. Thus, substantial capital resources are required for a company to consider installing a FMS. Smaller companies with restricted capital resources who want to build a FMS might first invest in a FMC . This approach can reduce the initial capital outlay and enable the earnings generated from the initial phases to fund later enhancements. A shorter initial phase for building a FMC can also give shorter commissioning time and shorter learning curves for company personnel.

The modular character of a FMC should be emphasized because : 1) an independent operable FMC module can be a basic component of a FMS, 2) a FMS can be built by integrating several FMC modules, and 3) using a FMC module concept can make the maintenance, extension and upgrade of a FMS much easier. The FMC, used as a basic component of a future FMS, has been built by integrating several Distributed Control Systems (DCS) to achieve "centered management and operation distribution" [Pan, 1995].

The primary physical components of a FMC are: 1) conveyor system, 2) material handling robot , 3) machining center, 4) inspection probes and 5) computer control system.

2-2-1 Conveyor System

A conveyor system is used when material must be moved in relatively large quantities between specific locations over a fixed path. Most conveyor systems are powered to move the load along the pathways. Conveyors have the following attributes:

1. They are generally mechanized, and sometimes automated.
2. They are fixed-in-position to establish the paths.
3. They can be either floor mounted or overhead.
4. They are almost always limited to one directional flow of materials.
5. They generally move discrete loads, but certain types can be used to move bulk or continuous loads.
6. They can be used for either delivery-only or delivery-plus-storage of items.

The major types of conveyors are the following: 1) roller conveyors, 2) skate -wheel conveyors, 3) belt conveyors, 4) chain conveyors, 5) slat conveyors, 6) overhead trolley conveyors, 7) in-floor towline conveyors, and 8) cart-on track conveyors [Groover, 1987].

2-2-2 Material Handling Robot

A material handling robot moves material or parts from one location and orientation to another. To accomplish the transfer, the robot is equipped with a gripper type end effector which must be designed to handle the specific part or parts to be moved in the application. Included within this category are the following two cases: material transfer and machine loading and/or unloading.

In nearly all material handling applications, the parts must be presented to the robot in a known position and orientation. This requires some form of a material handling device to deliver the parts into the work cell in the defined position and orientation [Groover, 1987]. This research focuses more on

machine loading and unloading, centered on both loading of the raw workpiece and unloading of the finished part by the robot.

2-2-3 Machining Center

The Machining Center (MC) , developed in the late 1950's, is a machine tool capable of performing several different machining operations on a workpiece in one setup under program control [Groover, 1987]. Machining centers are classified as vertical or horizontal in reference to the orientation of the machine tool spindle. A vertical machining center has its spindle on a vertical axis relative to the worktable, and a horizontal machining center has its spindle on a horizontal axis . The machining center is capable of milling , drilling, reaming, tapping, boring , facing and similar operations. In addition, the typical characteristics of the NC machining center include the following:

1. Automatic tool changing (ATC) capability - Automatic tool changers are applied for both horizontal and vertical machining centers. They significantly reduce in-cycle dead-time compared with NC type machines or early machining centers, which required an operator to change tools manually at a program cycle stop [Joseph, 1988]. There are two main reasons for providing the increased tooling capacity. First, the more tools available on a machine, the more workpiece types it should be capable of machining and the greater the flexibility of the machine and the system. The second reason is related to an unmanned system operation, which users are increasingly planning for. Some tools may require duplicates in a MC if tool wear is expected. If this occurs, another identical tool could replace the worn out tool without program stop and set-up procedure.
2. Automatic workpiece positioning - Most machining centers are capable of rotating the job relative to the spindle, thereby permitting the cutting tool to access four surfaces of the part.
3. Pallet shuttle - A MC could have two or more separate pallets that could be presented to the cutting tool.

2-2-4 Computer Control System

Although this research only focuses on a FMC not a FMS, when discussing a computer control system for FMC, the computer control system for the FMS must be described as well. This results from the fact that the role of computer control in the FMC already belongs to the top level of the FMS system, and because the controller of FMC will eventually be connected to a FMS. Networking or other linking methods will be used to integrate all the computer control systems of FMC modules together as a FMS.

The operation of a flexible manufacturing system or a complex flexible manufacturing cell is always computer controlled. Whereas earlier automation endeavors concentrated on the improvement of machining operations, at present attention is focused on 95% nonproductive moving and waiting time. As a matter of fact, most of the present research effort in manufacturing is concerned with the possibility of reducing this idle time, thereby increasing machine utilization and productivity. Since the task is very difficult to perform with conventional automation tools, the computer plays an ever-increasing role [Rembold, 1985].

Computer Functions

The functions performed by the FMS or FMC computer control system can be grouped into the following categories.

1. Control of each workstation of the FMC- This is required for individual processing or assembly stations.
2. Distribution of control instructions to workstations- Some form of central intelligence is also required to coordinate the processing at the individual stations. For example, in a machining FMC the working part's programs must be downloaded to the machines.
3. Production control- This function includes decisions on parts mix and rate of input of the various parts into the system. These decision are based on data entered into the computer, such as the desired production rate per day for the various parts, number of raw workpieces available and number of applicable pallets.

4. Traffic control- The term traffic control refers to the regulation of the primary workpiece transport system which moves parts between work stations. This control is affected by the division of the transport system into zones. A zone is a section of the primary transport system (towline chain, conveyor, etc.) which is individually controlled by a computer. By allowing only one cart or pallet in a zone, the movement of each individual workpiece is controlled. The traffic controller operates the switches at branches and merging points, stops workpieces at machine tool loading points, and moves parts to operator load/unload stations.
5. Shuttle control- The shuttle control regulates the secondary part handling systems of each machine tool. Each shuttle system must be coordinated with the primary handling system and must be synchronized with the operations of the machine tool it serves.
6. Work handling system monitoring- The computer must monitor the status of each cart and/or pallet in the primary and secondary handling systems, as well as the status of each of the various workpiece types in the system.
7. Tool control- Monitoring and control of cutting tool status is an important feature of a FMS or FMC computer system. There are two aspects of tool control: accounting for the location of each and tool-life monitoring.
8. System performance monitoring and reporting- The FMS or FMC computer can be programmed to generate various reports on systems performance that are desired by management.

2-2-5 Inspection Probes

Inspection probes are usually used in both Coordinate Measuring Machines(CMM) and machining centers to measure a workpiece in off-line and on-line modes, respectively. In recent years there has been significant growth in the use of tactile probes as on-line inspection systems in machining center applications. These probes are mounted on tool holders, inserted into the machine tool spindle, stored in the tool drum, and handled by the automatic tool changer in the same way that cutting tools are exchanged. When mounted in the spindle, the machine tool is controlled very much like a CMM. Sensors in the probe determine when contact has been established with the part surface.

The most common probes today are "touch-trigger" probes, which use a highly sensitive electrical contact that emits a signal when the end of the probe is deflected in the slightest amount from its neutral position. Immediately upon contact, the coordinate positions of the probe are recorded by its controller. After the probe has been separated from the contact surface, it returns to the neutral position. The probe is not merely positioned relative to the part, but its location also can be accurately and precisely recorded to obtain dimensional data concerning the part geometry.

This on-line inspection process can be employed to detect the zero reference point for a workpiece before cutting and immediately following the machining operation to detect the desired kind of measurements. These probes are used between machining steps in the same setup as for example, when establishing a datum reference either before or after initial machining so that subsequent cuts can be accomplished with greater accuracy.

An inspection probe for measuring workpieces in a machining center has many advantages over manual inspection methods. The principal advantages are:

1. Productivity - Because of the automated measuring techniques included in the operation of a machining center, inspection speed and labor productivity is improved. A machining center with a mounted probe and with operating functions like CMM is capable of accomplishing many measuring tasks including dimensions, hole location and diameter.
2. Flexibility - An inspection probe , which is mounted in the spindle of a machining center like the CMM, can be used to inspect a variety of different part configurations with minimal changeover time and it does this even better than the CMM because it does not need another setup procedure.
3. Reduce operator error - Automating the inspection procedure has the obvious effect of reducing human errors in measurements and setups.
4. Greater inherent accuracy and precision - An inspection probe in a MC is inherently more accurate and precise than the manual surface plate methods of inspection traditionally used.

2-3 Distributed Control System and Communication System

Distributed Control Systems (DCS) play an important role in modern factory automation because they can implement concurrent manufacturing and have flexible capabilities for FMC, FMS or CIM. Currently, most control systems for manufacturing either belong to the DCS category or are compatible with DCS. However, successful implementation of DCS depends on an effective communication system. In the following sections, both DCS and the communication for industrial automation are discussed.

2-3-1 Distributed Control System

Traditional automation was based on having mainframes or mini-computers to do centralized process control. However, because of the advances in semi-conductors and industrial process control devices, computer hardware and software advances have been significant. The result has been a shift to DCSs based on workstations or personal computers. This means more flexibility, better user interface, and open architecture for both networking and database [Pan, 1994]. The basic concept of the DCS is to use one or more computers to accomplish hierarchical control system's application via computer networking or other handshaking interfacing.

The basic component of DCS is a micro-computer based Process Control Station (PCS). Basically, a PCS has a Central Processing Unit (CPU), Read-Only Memory (ROM) , Random-Access Memory (RAM), and a Timer. These components are the control program executing unit. A PCS has analog and digital input and output, and it may also have a communication interface. The PCS's CPU uses an internal bus to communicate with analog and digital input and output units. A PCS's structure is shown in Figure 2-1.

A Distributed Control System consists of PCSs (at least one PCS); these PCSs use networking to pass messages between DCSs. There are two ways to construct a DCS:

- First, which is the most common, is the customized system. It is designed so that it has its own specialized standard devices. An example would be a simple PCS structure for a Puma 762 robot's PCS and a Machining Center PCS designed for a specific application.
- Second, is to use computers such as IBM compatible PCs, industrial computers, or along with appropriate interface cards for input and output functions (such as RS232 card or Analog/Digital (AD) card) to create a complete Process Control Station, such as the control computer of the conveyor system employed in this research.

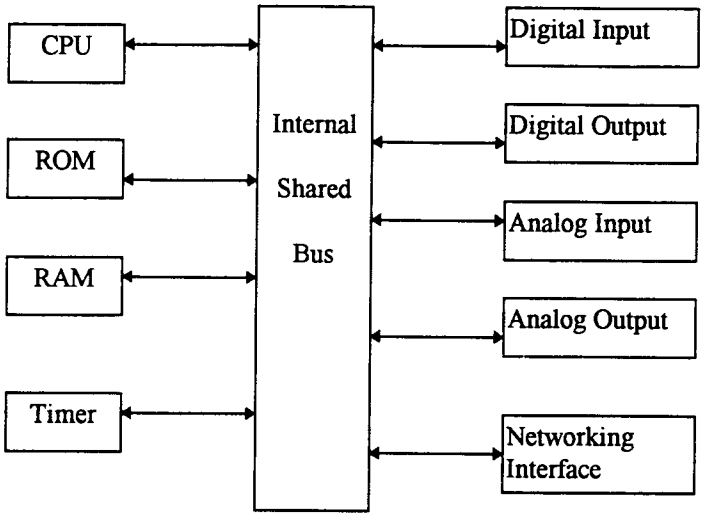


Figure 2-1 Process Control Station's Structure [Pan, 1995]

The future process control system, as described by Yei [1995], will include an information system. Yei also discussed the current development of the field bus, WorldFib, and technology of Programmable Logical Control (PLC). To integrate several DCSs into a desired manufacturing system, the communication problem between these DCSs must be solved. Approaches that can link several different DCSs together as an integrated system have been implemented by MAP_BAS LAN [Nagakaw, 1989], local operating network [Tsang, 1994] and Field bus architecture [Damsker, 1991; Wang, 1994]. Finally, an integrated DCS should be verified. In testing DCSs, Andersen [1991] described methods for

designing a simulator to test the control system of integrated DCSs for hardware, software and synchronization of components.

2-3-2 Communication System for Industrial Automation

A communication system in manufacturing is essential for effective use of the methodology broadly grouped under the generic term CIM. Included within this methodological group is the FMC. The factory communication system is the glue that joins together the various component parts of an overall manufacturing control system.

2-3-2-1 The Architectural Layers of CIM

A CIM system can be decomposed into five levels [Rodd et al, 1989], as shown in Figure 2-2.

1. The first layer is the information-process layer. This typically handles large-machine data processing type tasks; functions include materials control, order processing, financial control systems, production planning , CAD/CAM, etc.
2. The second layer , called the plant layer, is concerned with anything that is plant-wide in terms of scope. This layer includes management of facilities, production scheduling and order processing.
3. The third layer is the area manager level. It is concerned with procedures that are specific to a given area of the plant such as one complete production line.
4. The fourth level is the cell control layer. In many cases, a cell would be configured to produce one particular product at any one particular time and then reconfigured for an alternative product. A FMC can be viewed as a node of the forth layer with its branches in the fifth layer.
5. The fifth layer, the shop floor layer, covers the automation devices themselves including the controllers for robots, AGVs, machining centers and conveyors.

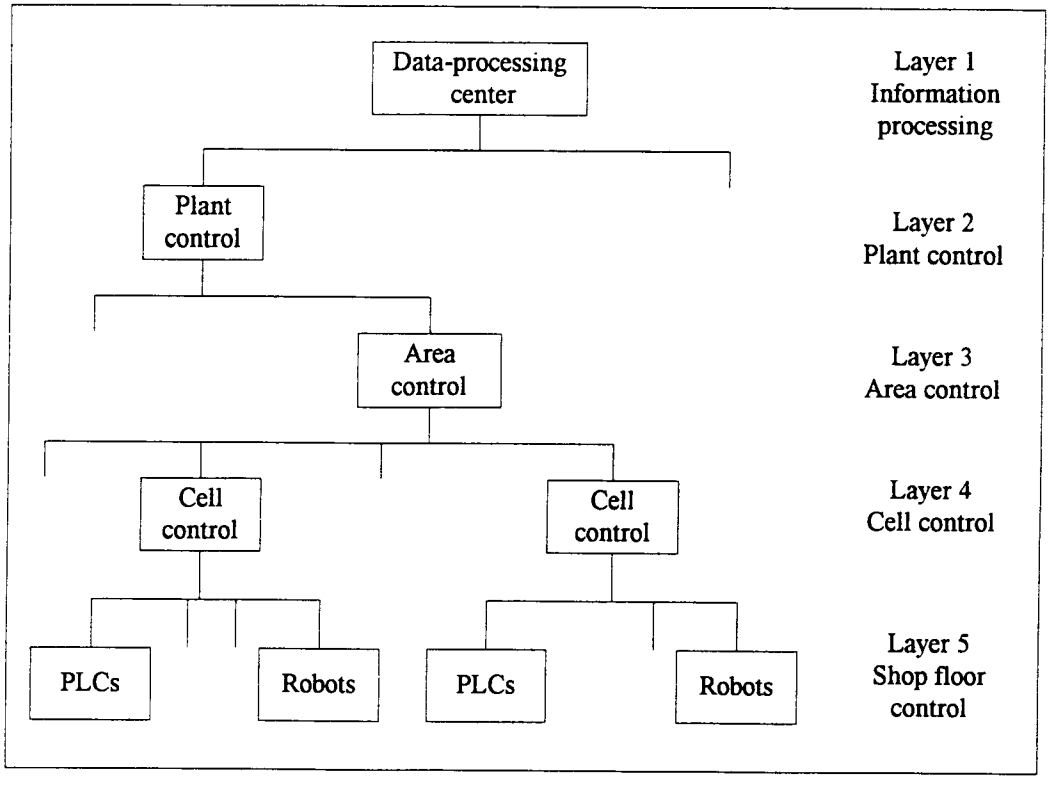


Figure 2-2 Architectural Layers [Rodd et al, 1989]

2-3-2-2 Communication System Architecture

International Standards Organization (ISO) Terminology

In the ISO model, each layer provides a service to the layer above and may use the services of the layer below (See Figures 2-3 and 2-4). An entity can be a software entity (such as a process), or a hardware entity (such as an intelligent I/O chip). The entities in layer N implement a service used by layer N+1. In this case, layer N is called the service provider and layer N+1 is called the service user. At the lowest layer (Figure 2-4), there is an actual physical communication between the machines of the network. Each layer of one machine carries on a virtual conversation with the corresponding layer of the other machines by applying rules and conventions collectively known as the N-layer protocol (Figure 2-4).

The general description of seven layers is as follows:

1. Physical layer - This layer is concerned with the transmission of raw bits across the network lines. It defines the data transmission rate and type of transmission medium.
2. Data link layer- This layer is concerned with the transfer of units of data across the local area network. This layer deals with the resolution of contentions when two devices are attempting to transmit message.
3. Network layer - The network layer is concerned with controlling the operation of the subnet. A key design issue is determining how packets are routed from source to destination. It stores and relays data traveling between the nodes in the network as part of this function.
4. Transport layer - The basic function of this layer is to accept data from the session layer, split it up into smaller units if need be, pass these to network layer, and ensure that the pieces all arrive correctly at other end.
5. Session layer - The session layer allows users on different machines to establish a session between them. It deals with network security issues, resyncronizing the data in the event of a transmission failure, and similar problems.
6. Presentation layer - This layer is concerned with negotiating syntax and format for the data exchange between the sending and receiving devices.
7. Application layer - This layer provides the interface with the user for specific applications requiring the networking capabilities. These applications deal with problems such as transfer of files between devices, remote job entry, message handling and access of files located at one device from another device.

The communication system of this research only focuses on layers 1, 2 and 7 because of the FMC control mechanism requirements.

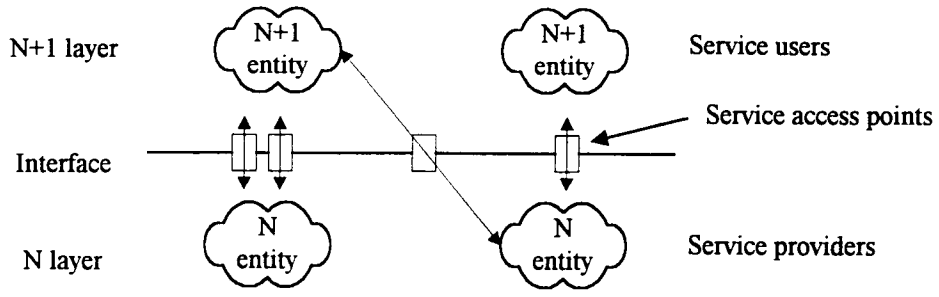


Figure 2-3 Service Access Points [Rodd et al, 1989]

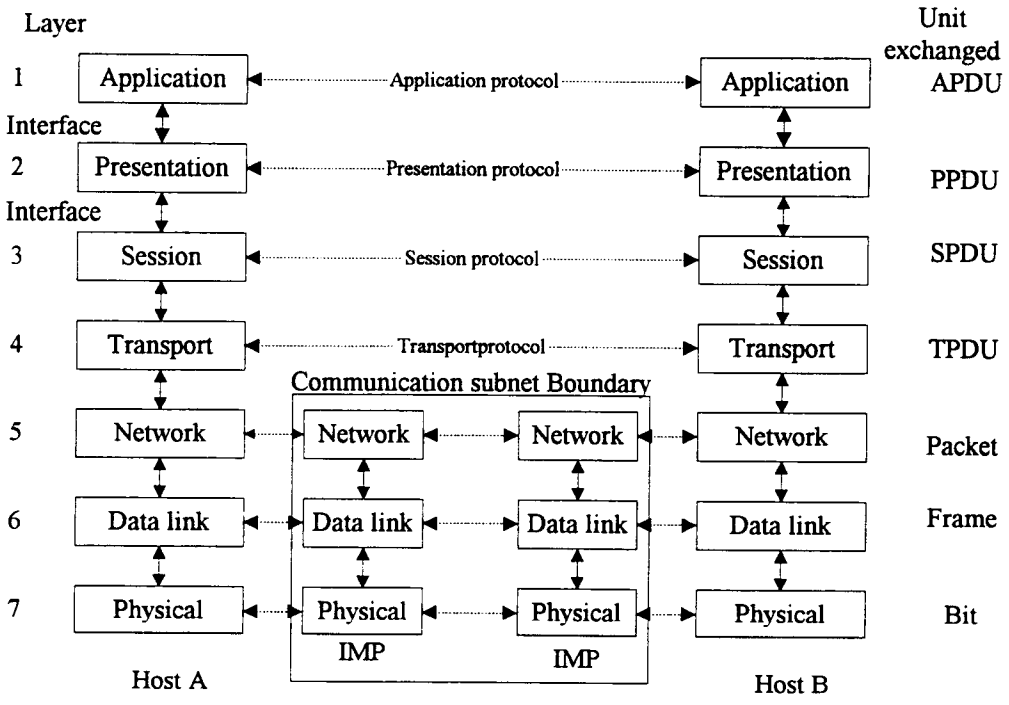


Figure 2-4 OSI Reference Model [Tanenbaum, 1991]

Connectionless Services And Connection-Oriented Services

Connectionless services provided in each layer do not enhance the reliability of the underlying service provided by other layers. These services are not able to prevent the loss, duplication or out-of-sequence delivery of messages. Like a telegram, each message includes the content and the full address of the destination without the need of any acknowledgment. In traditional factory automation, signals between a controlled system, sensors, and actuators belong to this category.

In this research, the signal transmission between the control computer and the other systems uses the connection-oriented service philosophy. Connection-oriented services are more complex than the connectionless ones because of the overheads involved. The connection-oriented services generally have three phases of operation, as shown in Figure 2-5. The full address of the destination must be given when a connection is established. The address of the resource is provided with connection indication. The connection-oriented service primitives [Tanznbaum, 1991] are the following:

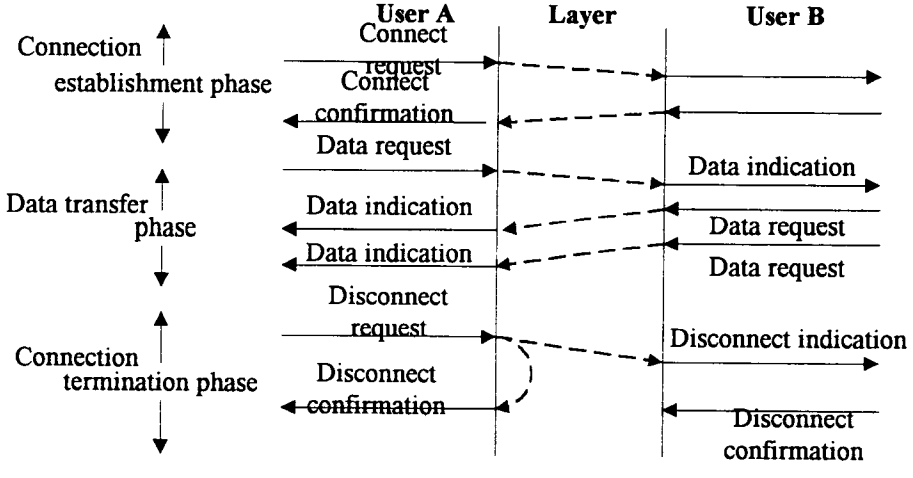


Figure 2-5 Connection-oriented Service [Sloman and Kramer, 1987]

1. CONNECT. request : Request a connection to be established

2. CONNECT. indication : Signal the called party.
3. CONNECT.response : Used by the caller to accept or reject calls.
4. CONNECT.confirmation : Tell the caller whether the call was accepted.
5. DATA.request : Request that data be sent.
6. DATA.indication : Signal the arrival of data.
7. DISCONNECT.request : Request that the connection be released.
8. DISCONNECT.indication : Signal the peer about the request.

2-3-2-3 Communication System Structure

There are two kinds of designs for the communication subnet:

1. Point-to-point channels. In this type of channel, there are many cables or telephone lines, each one connecting a pair of Interface Message Processors (IMPs). If two IMPs that do not share a cable but nevertheless wish to communicate, they must do this indirectly, via other IMPs. Figure 2-6 shows several possible topologies.

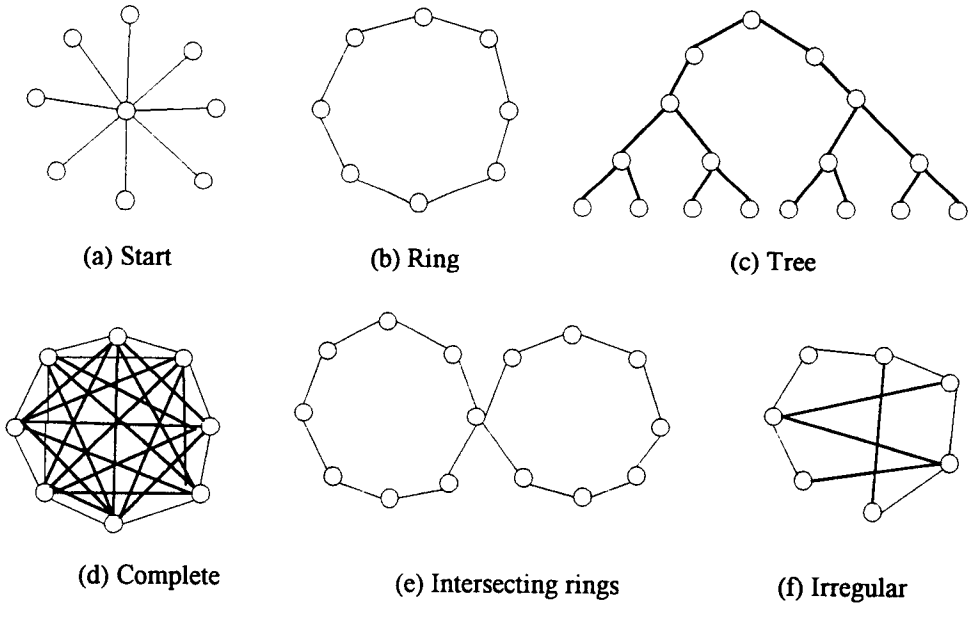


Figure 2-6 Some Possible Topologies for a Point-to-point Subnet [Tanenbaum, 1989]

2. Broadcast channels. Most local area networks and a small number of wide area networks are of this type. In a local area network, the IMP is reduced to a single chip embedded inside the host providing one host per IMP, whereas in a wide area network there may be many hosts per IMP. Broadcast systems (See Figure 2-7) have a single communication channel that is shared by all the machines on the network . Messages (packets) sent by any machine are received by all the others. Upon receiving a packet, a machine checks the address field. If the packet is intended for some other machine, it is just ignored.

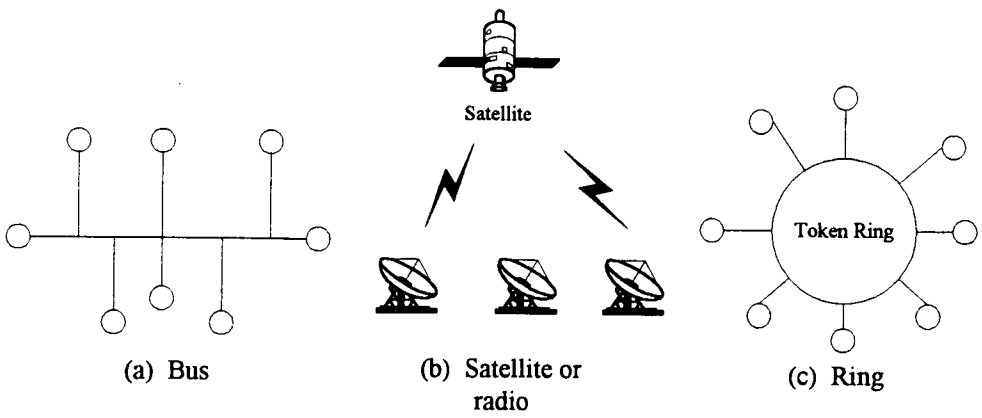


Figure 2-7 Communication Subnets Using Broadcasting

The details of communication system architecture and structure can be found in Tanenbaum [1989] and Rodd et al [1989].

2-3-2-4 Communication System in Manufacturing

In considering the implementation of a FMC, the real-time nature of the processes under control has to be considered as a fundamental design parameter of the communication system. The communication design in manufacturing should be able to cope with the worst possible scenario, that of

multiple plant failures. Therefore, the calculation of average delay through networks is of little significance. What is required is an analysis of the delays under worst case conditions.

For example, Ethernet works by having all the machines listen to the cable. If the cable is idle, any machine may transmit. If two machines transmit at the same time, there is a collision, in which case they all stop, wait for a random period of time, and try again later. In theory , there is no upper bound on the time a machine might have to wait to send a message. Therefore, protocols or hardware, such as the Technical and Office Protocols and Ethernet, are not suitable for factory automation.

There are three common communication approaches in manufacturing.

1. Interlock : Interlocks are a means of interfacing manufacturing devices that regulate the sequence of the program and coordinate the activities in the cell. The interface permits control signals to flow back and forth between the controller and the external device.

   For communication, this is a primitive approach. Interlock usually is implemented by OPTO 22 Solid State Relays (SSR), but its speed is slow. However, through interlock a controller can control sensors or actuators directly. By using input/output connections, an interlock can synchronize operations of other machines .

2. Manufacturing Automation Protocols (MAP) : Initiated by GM, MAP mainly covers the control level with the computers located in the manufacturing area. The MAP concept is based on a backbone like network which links single automation cells with each other. MAP follows the OSI model, using the token bus (IEEE 802.4) for the physical medium.

3. Fieldbus : For communication of and with sensors or actuators, the lower level of the system hierarchy is implemented via Fieldbus. In the manufacturing area, the Fieldbus forms a downward enlargement within the system hierarchy of the application area of MAP. The Fieldbus has the task

of establishing open communication between elements of the so-called peripheral microelectronics at installations and machines (sensors, actuators, measuring transducers, drivers, etc.), and those at an intermediate level, such as programmable logic controllers. Fieldbus protocols definition are based on layers 1, 2 and 7 of the ISO/OSI reference model.

2-4 Statistical Process Control

In Statistical Process Control (SPC), inferences are made concerning the quality of an item (such as product, subassemblies, components, or material) based on samples taken from the population of the items. SPC is a measure that attempts to address the noise in the manufacturing process, eliminating sources of waste and inefficiency that produce variation in the function of the product [DeVour et al, 1991]. SPC can improve quality and can limit waste and rework; in another words, it can improve productivity.

2-4-1 Manufacturing Process Variation

Shewhart described the variations in a process as arising from either chance causes or assignable causes. Assignable causes may be thought of as problems that arise periodically in a somewhat unpredictable fashion and can usually be dealt with effectively at the machine or workstation level. For example, broken tools, a jammed machine, material contamination, machine-setting drift, operator error and defective raw material all belong to the class of assignable causes.

The ever present chance causes constitute problems with the manufacturing system itself, influencing production until found and removed. The most common chance causes include poor supervision, poor training, inappropriate methods and poor workstation design. These problems can be generally specified and change implemented at the management level.

2-4-2 Automated Inspection Method

If SPC inspection and testing are carried out manually, a process is stable, or sampling is too expensive and time consuming, then the sample size is often small compared to the size of the population. The sample size may only represent 1% or less of the number of parts made in a high-production run. In principle, the only way to achieve 100% good quality is to use 100% inspection. However, 100% inspection using manual methods is no guarantee of a 100% good quality product because it may contain inherent errors due to the measurement procedure, or mistakes may result from operator's fatigue or other factors. The 100 % automated inspection process offers an opportunity to overcome the problems associated with manual inspection.

The timing of the inspection procedure that relates to the manufacturing process is an important factor in quality control. There are three different kinds of inspection as follows:

1. Off-line inspection - An off-line inspection is performed after the manufacturing process is complete, introducing a time delay between processing and inspection. Most manual inspection belongs to this category. This kind of inspection is risky because parts will have already been made by the time any poor quality is detected.
2. On-line/in-process inspection - An on-line/in-process inspection is performed during the manufacturing operation. The inspection measures specified dimensions at the same time the parts are being made . It is possible to influence the operation that is processing the current part. As a result, a potential quality problem can be fixed before the part is completed. However, technologically automated, on-line/in-process inspection is usually difficult and expensive to implement.
3. On-line/postprocess inspection - An On-line/postprocess inspection is performed by measuring or gauging the parts immediately following the production process. Unlike the on-line/in-process inspection method, the on-line/postprocess inspection method is easier to accomplish. Although this inspection can not effect the part that has been made, it can influence the manufacturing process of the next part.

An automated manufacturing system may have distributed inspection with several inspection stations located along the line of work flow. A distributed inspection system should have the capability to detect out-of-control signals in a process or any defective product as soon as possible in order to take the necessary actions to respond to quality problems. This distributed inspection strategy is used to prevent further costs being added to defective products.

2-4-3 Control Charts (X bar And R charts)

Control charts are the tools to implement the SPC's observation, evaluation and diagnosis. They also help in making decisions to improve out-of-control processes. There are two fundamental purposes of control charts: [DeVour et al, 1991]

1. They assist in identification of both chance and assignable faults in the process and help provide the basis to formulate improvement action for off-line activities.
2. They provide an economic basis for making a decision "at the machine" as to whether to look for problems and adjust the process or to leave the process alone.

A control chart is a technique for plotting the measured values of certain characteristics of the process output over time to determine if the process remains in statistical control. In this research, an $\overline{X}$ control chart and an R control chart are employed. The $\overline{X}$ chart is used to plot the average measured value of a certain quality characteristic for each of a series of samples which are taken from the production process. The $\overline{X}$ chart indicates how the process mean varies over time. In the R chart, the range of each sample is plotted. It monitors the variability of the process and indicates whether the variability changes over time.

$\overline{X}$ And R Chart Set Up Procedure:

- Sample selection - The first step in setting up an $\overline{X}$ and an R chart is the selection of sample size and frequency. All the samples must be rational samples (for details on rational sampling method please refer to DeVour et al, 1991).
- Initiation of the control charts - To provide a solid basis for the initiation of the control chart, 25 to 50 samples should be selected. The central lines and control limits of the control charts are calculated in this phase.

1. For each sample, an average of sample *i* , $\overline{X}$, is calculated as:

$$\overline{X}_i = \frac{\sum_{j=1}^{n} X_{ij}}{n}, \qquad (2.1)$$

where n is the sample size, X_{ij} is the *j*th measurement (j=1,...,n) in the *i*th sample (i=1,...,k).

2. The spread within *i*th sample is measured by the range R_i that is usually employed as a measure of within-sample variability.

$$R_i = X_{largest} - X_{smallest} \qquad (2.2)$$

3. The grand average, $\overline{\overline{X}}$, is the arithmetic average of all available sample averages. This grand average is an estimate of the process mean μ_x and becomes the centerline of the $\overline{X}$ control chart:

$$\overline{\overline{X}} = \frac{\sum_{i=1}^{k} \overline{X}_i}{k} \qquad (2.3)$$

where k is the number of samples being used to setup the control chart.

4. The average of the sample range is $\overline{R}$ which is the center line of the R chart is computed from:

$$\overline{R} = \frac{\sum_{i=1}^{k} R_i}{k} \qquad (2.4)$$

5. The upper and lower control limits of the $\overline{X}$ control chart are:

$$UCL_{\bar{x}} = \overline{\overline{X}} + A_2\overline{R} \qquad (2.5)$$
$$LCL_{\bar{x}} = \overline{\overline{X}} - A_2\overline{R}$$

The value of A_2 for varying sample sizes n can be found in quality control texts; see for example DeVour et al [1991].

6. The control limits of the R control chart are:

$$UCL_R = D_4\overline{R}\ ,\ LCL_R = D_3\overline{R} \qquad (2.6)$$

The values for D_3 and D_4 are a function of the sample size n (see DeVour et al, 1991).

- Plot the control charts. Once the center lines and control limits for the $\overline{X}$ and R charts are calculated using the formulas (2.1) to (2.6), $\overline{X}$ and R from successive samples are plotted on the control charts.

Interpretation Of $\overline{X}$ Chart And R Chart:

Nelson's [1984] eight tests are applied in this research to interpret both control charts. These tests provide the basis for the statistical signals which indicate that the process has undergone a change in its mean level, variability level, or both. To perform these tests, the $\overline{X}$ control chart is divided into 3 different zones A, B and C. These zones are between upper and lower control limits and are defined by 1, 2, and 3 sigma boundaries, as shown in Figure 2-8.

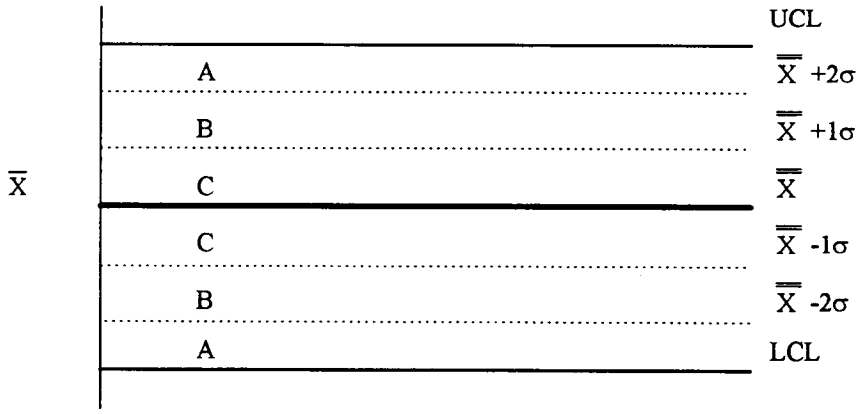


Figure 2-8 $\overline{X}$ Control Chart Zones to Aid Chart Interpretation

The eight test rules are:

For $\overline{X}$ Charts:

1. Test 1: Extreme Points - The existence of a single point beyond a control limit signals the presence of an out of control condition.
2. Test 2: Two out of three points in zone A or beyond - The existence of two out of any three successive points in zone A or beyond signals the presence of an out-of-control condition.
3. Test 3: Four out of five points in zone B or beyond - The existence of four out of any five successive points in zone B or beyond signals the presence of an out-of-control condition.
4. Test 4: Run above or below the centerline - When there are eight or more successive points either strictly above or strictly below the centerline, there is a strong indication that the process has shifted from the centerline.
5. Test 5: Linear trend identification - When six successive points on the $\overline{X}$ chart show a continuing increase or decrease, a systematic trend in the process is signaled.
6. Test 6: Oscillatory trend identification (in the spirit of test 5) - When fourteen successive points oscillate up and down on the $\overline{X}$, a systematic trend in the process is signaled.
7. Test 7: Avoidance of zone C test - When eight successive points occurring on both sides of the centerline avoid zone C, an out-of-control condition is signaled. The reasons for such a condition might be: 1) the occurrence of more than one process being charted on a single chart, 2) perhaps over control of the process, or 3) it may also be indicating that improper sampling techniques are begin used, particularly , process mixing.
8. Test 8: Run in zone C test - When 15 successive points fall on either side of the center line in zone C only, an out-of-control condition is signaled. The reasons for such a condition might be: 1) improper sampling, or 2) a change in process variability that has not been properly accounted for in the $\overline{X}$ chart control limits.

For R Charts:

1. Test 1: Extreme Points
2. Test 4: Run Above or Below the Centerline
3. Test 5: Linear Trend Identification
4. Test 6: Oscillatory Trend Identification

2-4-4 Real-Time SPC Applications

A real-time SPC application must implement the on-line inspections, analyze collected sample data plus diagnose and make decision in out-of-control conditions in a short period of time. There are not many real-time SPC applications reported in published literature. Lee [1991] discussed his experience with an on-line SPC system showing how to use SPC methods to determine real-time process alarm limits in a distributed monitor/control system. Mamzic et al [1991] provided a tutorial on SPC for manufacturing and described the implementation of SPC functions within a distributed control system. A conceptual, real-time SPC software structure that included operator advisories and an expert system has been introduced by Yeager and Davis [1992]. Noaker [1995] describes several requirements of a real-time SPC application including links to other components of the system. Finally, Marsh and Tucker [1989] reviewed SPC techniques that can be applied to batch processes

2-5 Petri Nets for Modeling Systems Integration

The Petri net is a formal, graphic representation technique. It is a systematic method that has been developed for synthesis and analysis. Petri nets are particularly suited to model distributed and concurrent systems, such as communication systems, computer software, computer hardware and manufacturing processing, because Petri nets exhibit synchronization, mutual exclusion and cooperation among concurrent operations. Silva et al [1993] claimed that Petri net theory is the only tool which can support not only the specification activity, but also the evaluation of the behavior of the system starting from the model provided by the specification.

Petri nets can be used in designing the operation of a manufacturing cell [Teng, 1991]. They have been used in modeling an automated storage/retrieval system with the characteristics of concurrence, conflict and deadlock [Knapp, 1992]. Ferrarini and Mafferroni[1991] proposed a conceptual framework based on special version of Petri nets for designing logic controllers. Tsai et al [1995] presented time constraint Petri nets and described how to use them to model real-time system specifications and to determine whether the specifications can be scheduled with respect to imposed timing constraints.

Petri net based tools have been developed to simulate manufacturing systems. Yim and Barta [1994] developed a modeling methodology and a tool for the simulation of FMSs. Capkovic [1991] developed a suitable general extension methodology, from the system theory perspective, that focused on the possibility of an analytical formulation of the problem concerning the control synthesis for discrete events dynamic systems.

Petri nets were invented in the 1960's by Carl Petri at the University of Bonn, Germany and have been popular in Europe. The theory of Petri nets has been developed by a number of people at different times, and places with different backgrounds and motivations. Therefore, there are many alternative forms for defining Petri nets. The various definitions of J. L. Peterson [1981], Hack [1975] and Silva [1993] are employed in this research.

Basic Petri Net Structure

A Petri net structure, C, is a four-tuple, C={P, T, F, B}, where

- $P= \{p_1, p_2, \ldots\ldots p_n\}$ is a finite set of places, $n>=0$.
- $T= \{ t_1, t_2, \ldots\ldots t_m\}$ is a finite set of transitions, $m>=0$. The set of places and the set of transitions are disjoint, $P \cap T = \varnothing$.

- $F \subseteq P \times T$ is a set of input arcs. $B \subseteq T \times P$ is a set of output arcs. F and B are functions that map places and transitions onto the number of tokens needed for input (F) or produced for output (B).

A place P_i is an input place of a transition t_j if $p_i \in I(t_j)$; P_i is an output place of a transition t_j if $p_i \in O(t_j)$ (the input function I and output function O are explained latter). In a manufacturing system, a place is a condition and a transition is an event. For example, in an AS/RS, a place represents the state of condition (e.g. a part in position) or the availability of a resource (e.g. a material handling robot). Events are actions which take place in the manufacturing system. The occurrence of these events is controlled by the state of the system. The state of the system can be described as a set of conditions [Peterson, 1981]. For example, when both part in position and material handling robot are available (places) are true, the event, the robot moves the part (transition t_1), is enabled (see Figure 2-9).

Input And Output Function

I: $T \rightarrow P^{\infty}$ is the input function, a mapping from transitions to bags (collection) of places.

O: $T \rightarrow P^{\infty}$ is the output function, a mapping from transitions to bags of places.

The input and output functions relate transitions and places. The input function I is a mapping from a transition t_j to a collection of places $I(t_j)$, known as the input places of the transition. The output function O maps a transition t_j to a collection of places $O(t_j)$, known as the output places of the transition.

Token in Petri net

A token is a primitive concept for the Petri net. Tokens are assigned to and are thought to reside in the places of a Petri net. The number and position of tokens may change during the execution of a Petri net.

Petri Net Marking

- A marking μ is an assignment of tokens to the places of a Petri net, C= {P, T, F, B}. It is a function from the set of places P to nonnegative integers N: μ: P→N.
- A marking μ is also an n-vector $\mu = \{\mu_1, \mu_2, \ldots\ldots \mu_n\}$, where n= | P | and each $\mu_i \in N$, i = 1,...n. The vector μ gives for each place p_i in a Petri net the number of tokens in that place; μ_i represents the number of tokens in place p_i.
- The definitions of marking as a vector are obviously related by $\mu(p_i)=\mu_i$.

Marked Petri net

A marked Petri net M= {C, μ} is a Petri net structure C= {P, T, F, B} and a marking μ. This is also written as M= {P, T, F, B, μ }.

Enabled Transition

A transition is enabled if $\mu(p) > 0$ for all $p \in I(t)$.

Firing Rule

A firing rule has two aspects: 1) a transition is enabled if every input place has at least one token and 2) when an enabled transition can fire, each input place of that transition loses one token. When a transition fires, the marking of places p changes from $\mu(p)$ to $\mu`(p)$ as follows:

$\mu`(p)= \mu(p) + 1$ if $p \in O(t)$ and $p \notin I(t)$

$\mu`(p)= \mu(p) - 1$ if $p \notin O(t)$ and $p \in I(t)$

$\mu`(p)= \mu(p)$ otherwise

A transition firing represents the utilization of a resource [Silva et al, 1993]

Petri Net Graphs

A Petri net graph has two types of nodes and direct arcs. A circle O represents a place; a bar | or a box □ represents a transition. Directed arcs connect places (O) and transitions (| or □) to define their relationship. An example in Figure 2-9a and 2-9 b explains Petri net structure and graphics.

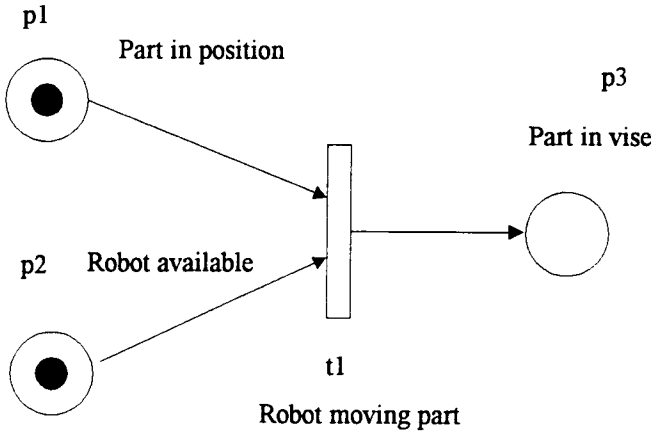


Figure 2-9a A Petri Net Structure before Firing

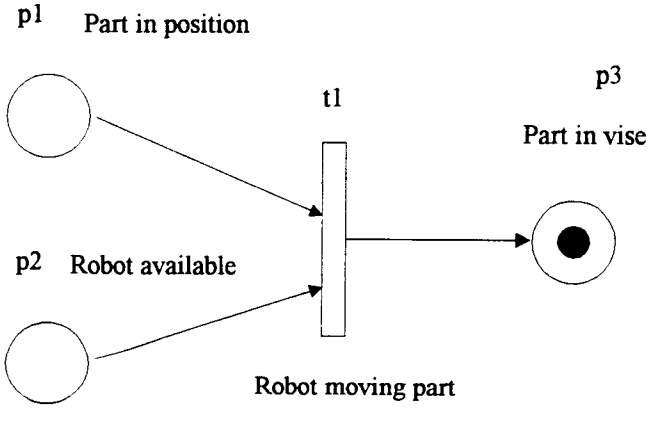


Figure 2-9b A Petri Net Structure after Firing

In Figure 2-9, Petri nets' structures are the following:

1. Input and output functions:

$I(t_1)=\{ p_1, p_2 \}$ $O(t_1)=\{ p_3 \}$

2. The Petri nets' structures before firing (Figure 2-9a):

$P=\{p_1, p_2, p_3\}$ $T=\{t_1\}$ $F=\{(p_1, t_1), (p_2, t_1)\}$ $B=\{(t_1, p_3)\}$ $\mu=\{1, 1, 0\}$

3. The Petri nets' structures after firing (Figure 2-9b):

The input functions, the output functions, and Petri nets' structures are the same, but the marked vector changes to $\mu=\{0, 0, 1\}$.

Petri nets can model the integrated manufacturing system architecture used in this research. Every manufacturing process can be put in either a place (condition) or a transition (action) of a Petri net model. Input and output functions can define a place mapping to a set of transitions, and a transition mapping to places, respectively. These functions define the logical relationship for an integrated manufacturing system specifically and graphically using input arcs (F) and output arcs (B) to connect places and transition. A token of a Petri net model represents a current manufacturing process flow or a material flow. Finally, with enable transition and firing rules, the Petri net can represent the dynamic run-time situation for an integrated manufacturing system among several DCSs. Petri nets are employed in modeling and system testing the integrated manufacturing process, and design the I/O bus in this research.

CHAPTER 3 INTEGRATION ARCHITECTURE

3-1 Overview

An integration framework for the general flexible manufacturing cell is shown in Figure 3-1. The framework consists of four phases: requirement analysis and specification, system design and modeling, system evaluation and implementation. The design of the framework is based on hardware and software specification of the physical components, manufacturing process requirements and functional requirements.

The physical components specification consists of completely and consistently specifying the DCSs' hardware and software characteristics used in the FMC. A primary manufacturing process requirement is interactive and cooperative operation among DCSs.

Functional requirements depend on the features described in the manufacturing system. Two of the more important considerations in many FMCs are Group Technology and real-time Statistical Process Control. The use of GT result in several benefits including: 1) reduction of part transportation times, 2) use of universal production equipment, 3) reduction of number of machining operations, 4) reduction of NC (or processing station's) programming time, 5) shorter production cycles, 6) reduction of setup and processing times, and 7) better control of manufacturing processes.

For a FMC, an effort is made to design group jigs and fixtures that will accommodate every member of a parts family. Workholding devices are designed to use special adapters which convert general fixture into one that can accepted by each part family member. The machining tools in a FMC do not require drastic changeovers in setup because of the similarity in the workparts processed on them. The FMC's material handling and retrieved systems also need to be designed to transport each part family member. For example, a pallet or a material handling robot's gripper can be designed to transport every part family member without incurring additional set ups.

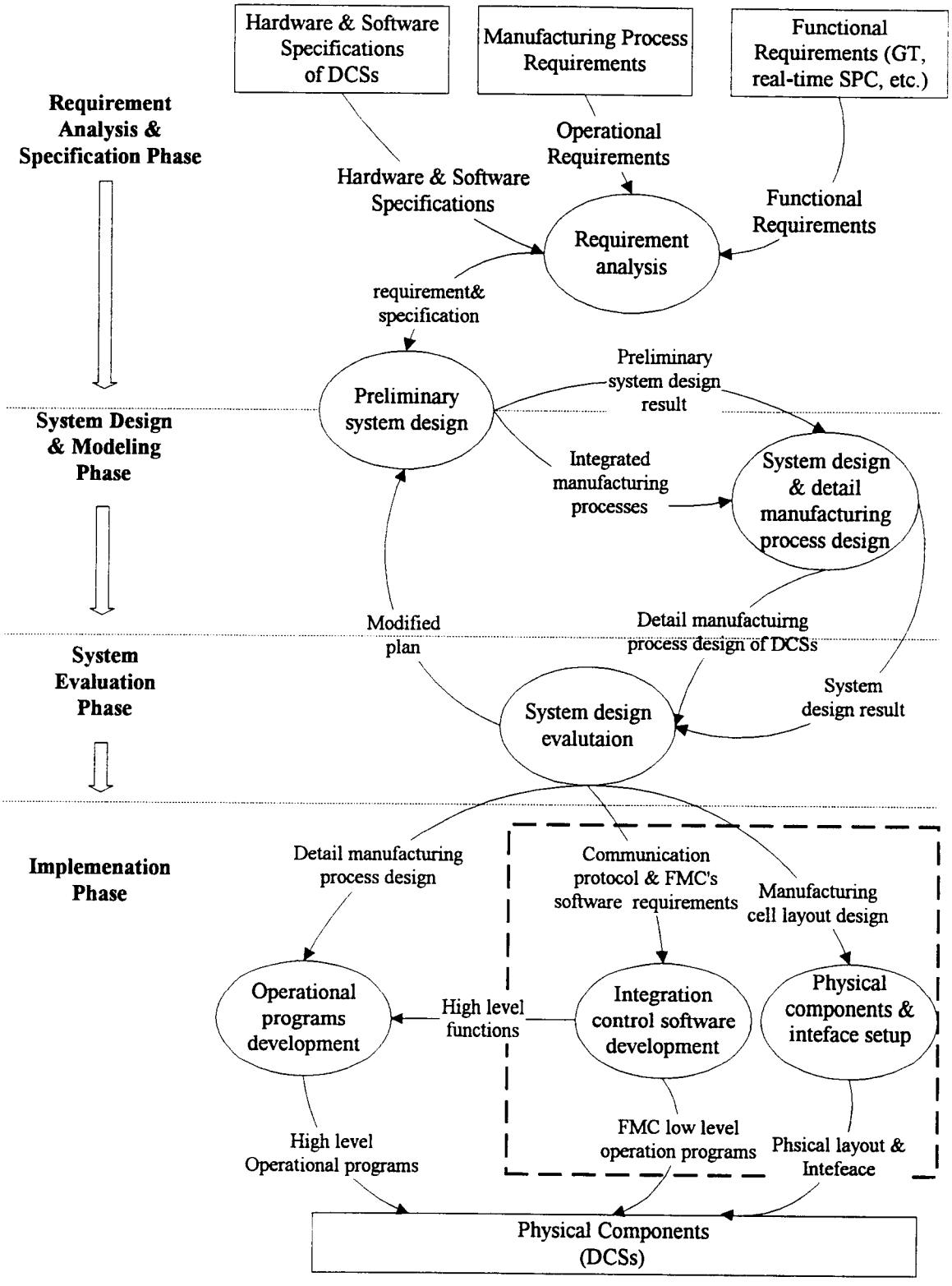


Figure 3-1 A Framework for FMC Integration

A second desired feature is real-time SPC capabilities. The real-time SPC system will perform data collection, data analysis and diagnosis immediately after a part is made. When an out-of-control signal is generated, the FMC's control computer will do the decision making and implement it in real-time. A real-time SPC framework is in Figure 3-2.

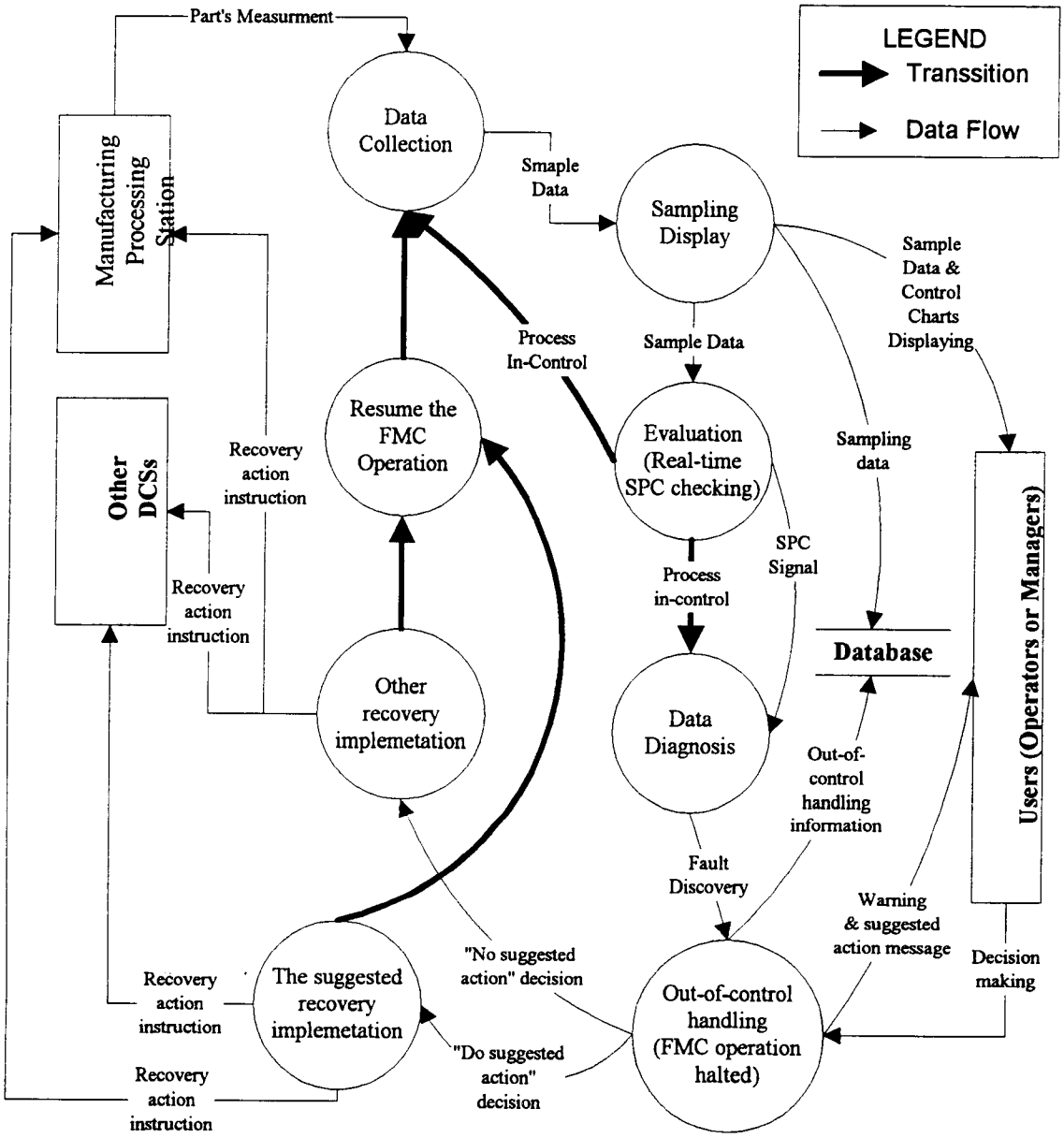


Figure 3-2 Real-Time SPC Framework for FMC

3-2 Integration Architecture

As mentioned in the previous section, the integration architecture consists of four phases: Requirement Analysis and Specification, System Design and Modeling, System Evaluation and Implementation.

3-2-1 Requirements Analysis and Specification Phase

In the requirement analysis phase, a FMC's requirements and specification must be clearly. These requirements include: 1) physical environment, 2) interfaces, 3) users and human factors, 4) functionality, 5) data, 6) resources, 7) safety, 8) quality assurance, and 9) flexibility.

It is important that the requirements be validated. There are seven criteria for requirement's validation (Pfleeger, 1987):

1. Are the requirements correct?
2. Are the requirements consistent?
3. Are the requirements complete?
4. Are the requirements realistic?
5. Are the requirements verifiable?
6. Are the requirements traceable?
7. Does each requirement describe something that is needed by the customer (the customer could be a manger, engineer, or experienced operator in the FMC system development)?

Several tools can be used to specify FMC's requirements including Data Flow Diagram (DFD), state transition diagram, natural language, Jackson Structure Programming (JSP) and Jackson System Development (JSD) and Scenario Integration Table. This research uses DFD and Scenario Integration Table as these can also be used in the system design phase.

The output from the requirement analysis phase is documentation for customer and system developer. Furthermore, the requirement must be organized in such a way that they can be tracked though

system development. The Scenario Integration Table can be used to represent the result from the requirements analysis phase.

Several tools are available for low-level control system development, such as each DCS's control system. These include: DFD, Jackson Structure Programming (JSP) / Jackson System Development (JSD), State Transition Diagram, Hierarchy and Input-Process-Output (HIPO), Input-Process-Output (IPO), Hierarchy Data Structure and Warnier Diagram. Other methods designed for software development include PSL/PSA, SREM, SAPT, SSA and Gist. These begin with a description of requirements in a formal language or structure so that system developer can analyze the requirements automatically for characteristics such as consistency and completeness.

3-2-2 System Design and Modeling Phase

A scenario integration table, used in system design and modeling phase, describes each distributed control systems' procedures and synchronization. To apply a scenario table in a FMC's integration, communication protocols and hardware, and other high level functions must be developed to support the linkages among DCSs and control commands communication. A second step in this phase translates this scenario to Petri net models. Mutually exclusive working parts must be defined, and synchronization must be verified to prevent deadlock and other system bottleneck.

3-2-2-1 Scenario Integration Table

An example of using a Scenario Integration Table is shown in Figure 3-3. The FMC in Figure 3-4 is based on a machining center, robot, conveyor and control computer. The control computer is in the first column of the table with each physical component or DCS in subsequent columns. Each row in the table represents the interaction between individual components. In general, synchronization is built between the control computer and its subordinate component operations using high level commands. For example, when a computer receives the pallet with raw material, it sends a message (e.g. "Arrival") to a waiting robot. This interaction is recorded in the table as shown in Figure 3-3 to show that

synchronization between the computer and the robot is to be established before successful completion of the subsequent operation.

Control Computer		Conveyor		Robot		Machining Center	
No	Operation	No	Operation	No	Operation	No	Operation
CC 12	Send an "Arrival" message to the robot Send(Robot, Arrival)			R6	Wait for an "Arrival" message Wait(Robot,Arrival)		
CC 13	Wait for an "Ack" message from the robot. Wait(Robot, Ack)			R7	Send an "Ack" message Send(Robot,Ack)		

Figure 3-3 An Integrated Manufacturing Process Scenario Table

Such commands can be formally be defined as "Send(station, command)" and "Wait (station, command)" functions between the components involved in the interaction. These function are powerful, convenient, flexible and easy tools to "glue" several DCSs together as a FMC (see Figure 3-4).

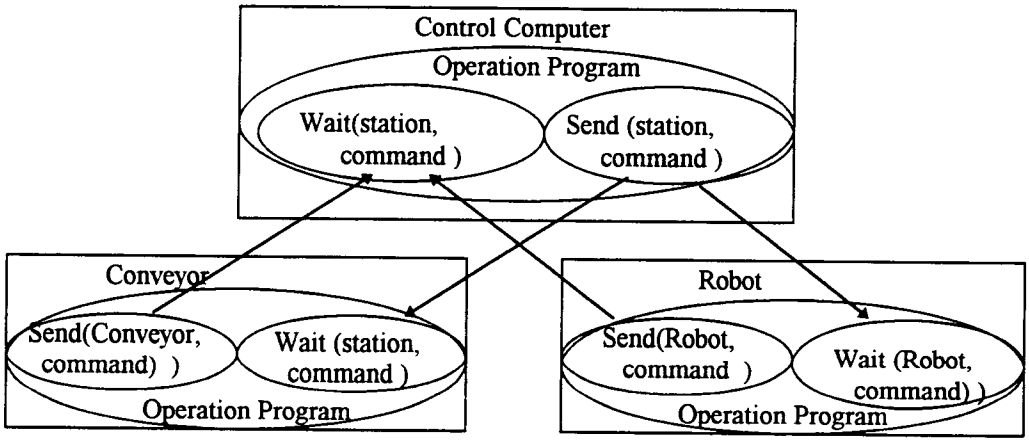


Figure 3-4 Linkages in a FMC

3-2-2-2 Petri Nets Modeling

Although integration scenario tables are easy to develop and understand, they are not a perfect tool for data flow analysis. The primary focus in scenario integration tables is on the computer as compared to individual DCS components. Logic relationships are not easy to represent in a scenario

integration table. Petri nets help to overcome these deficiencies. They can be built from an scenario integration table with ease because there is an almost one-on-one mapping relationship between an integration table and its Petri nets. Petri nets are used to model a FMC's concurrent execution and the interaction among DCSs, the parts' flow and current operation flow.

3-2-3 System Evaluation Phase

Petri nets are used in the system evaluation phase to analyze the manufacturing process's performance and utilization. Petri nets can also perform system design's testing to check concurrent execution, mutually exclusive situations and to prevent deadlock among components. Besides system evaluation, the Petri net's graphical presentation allows optimization analysis for each DCS. The scenario integration table might need to be modified after the system testing phase; however, it is valuable in the preliminary design and it is a good communication tool for users who do not understand Petri nets.

Specific considerations in the system evaluation phase include:

- Check operating sequence: Verify that the design system can operate the required FMC's operation procedures.
- Check for deadlock: Prevent the potential deadlock conditions which will cause the system to "halt" because there of a process waiting for a resource used elsewhere.
- Check for mutual exclusive condition: Check the required mutual exclusive implementation in system design, as example, a material handling robot being shared by two machining centers.
- Evaluate system utilization: Evaluate the designed operation procedure. Rearrange the operational procedure sequence if a DCS's waiting time can be reduced.

3-2-4 Implementation Phase

In the implementation phase, several high level operational programs are generated. These are bases on integrated manufacturing process scenarios and Petri nets models developed in earlier phases. Operational programs for each DCS are developed using the individual DCSs.

To illustrate, the physical system in this research includes a Puma robot for loading and unloading parts on machining center and a conveyor to transfer parts. A conveyor program is designed to transport pallets with working parts based on the conveyor Petri nets. Also, the conveyor operational programs need to handle many sensors and actuators, and perform traffic control. Similarly, Puma robot's operating program not only follows its Petri nets model, but also needs to consider gripper's closing and opening, working envelop and safety. The machining center's operating program template needs to be designed for manufacturing and probing procedures. This needs more detail design than Petri net model developed in the previous phase. Overall, the transformation between Petri net models to FMC's operational system is easy because the Petri net graphical structure is a highly structured representation for the equivalent logic relationship.

If the FMC's manufacturing cell layout design, interface, or the FMC's software requirements change, the FMC system must be modified to accommodate the change. This may only require change in a specific operation procedure. In most cases, the change may be a minor modification in the operational high level program for the associated DCS.

3-3 The Control Architecture for the FMC

In general, a FMC consists of several DCSs, actuators and sensors. The overall control architecture of the FMC system, represented as a Data Flow Diagrams (DFD), is shown in Figures 3-5. The external entities of this system are:

1. Users: They could be system developers, operators, or managers. The users use control computer system to run manufacturing production, monitor the DCS's operations and perform other tasks such as real-time SPC.
2. Automated Material Handling (AMH) system: The AMH system is used to transport working parts to the FMC's unloading location, and to transport finished parts to the next workstation. The AMH system can be conveyor or AGV; it can be shared by other FMC or processing stations.
3. Automated Material Retrieval (AMR) system: The AMR system in the FMC can move raw material from the AMH into the processing station, and move a finished part onto the AMH system. The AMH can be a material handling robot or other automated material feeder devices.
4. Processing station: The processing station is typically a machining center or other CNC that performs machining operation on families of parts. However, a FMC can be designed with other types of processing equipment, such as assembly workheads and sheet metal presses.
5. Other DCS entities: Besides the above basic DCS entities of a FMC, there might be other DCSs based on desired FMC's characteristics.
6. Sensors and Actuators: Besides DCSs' sensors and actuators, a FMC might have some sensors or actuators which are directly controlled by the control computer, as example, the fixture device and the sensor in a processing station.

Every DCS of the FMC has its control system for the operation and integration of the control computer . These DCSs' control systems are all connected to the control computer's control system. Data flow arcs link the external entities and the control systems. Dataflows' transmission between the computer's control system and the connected DCSs' control system is implemented by the communication interface. The communication interface requires easy set up and flexible messages transmission. The communication protocol and hardware can apply Field bus, MAP or I/O Bus.

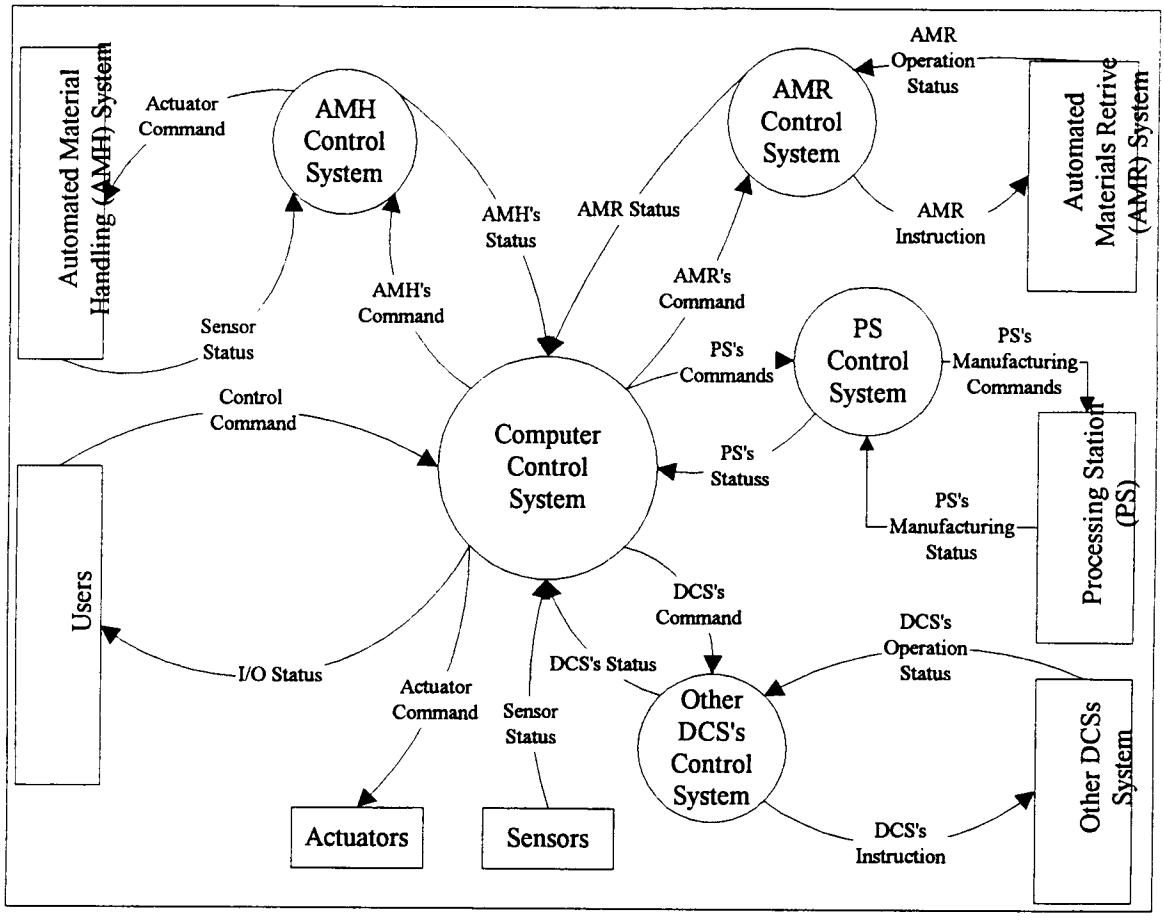


Figure 3-5 FMC Control Architecture Data Flow Diagram

CHAPTER 4 SYSTEM DESIGN AND IMPLEMENTATION

4-1 Overview

The integration architecture methodology in Chapter 3 is used to design and implement the FMC's integration and its requirements. This FMC's manufacturing system used in this research is described in Section 4-2. The complete description of the integration architecture's implementation is given in Section 4-3. The development of the FMC in this research is both to-down and bottom-up. For system development, the functional requirements are first defined and capabilities and limitations of the hardware and software for every DCS are identified. The next step is to process high level system design. However, the communication interface and other low level systems are designed and implemented concurrently while the high level system design is being developed. The system design and development is modular; each module is tested before proceeding to the next module.

This research uses Data Flow Diagrams (DFD) to identify external entities, transfer procedures, data flows and data storage. These DFDs can help develop the control software and a communication interface, and design an integrated manufacturing process for the FMC. A general description for the system in this research using DFDs is given in Section 4-4. Interfacing and messages passing methods are discussed in Section 4-5.

4-2 Manufacturing System Description

The integrated FMC used in this research consists of a conveyor system, a material handling robot, a machining center and associated control computers. The physical system is shown in Figure 4-1, and is located in the CIM manufacturing laboratory in the Department of Industrial and Manufacturing Engineering at Oregon State University.

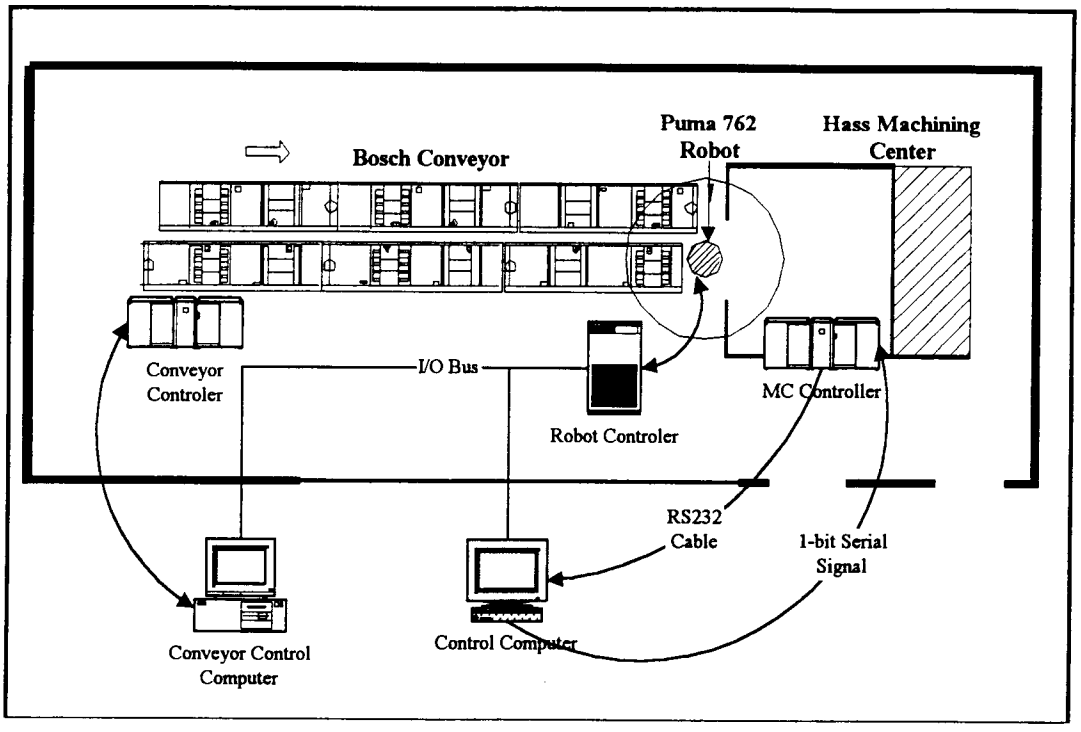


Figure 4-1 The Physical Layout of the Flexible Manufacturing Cell

The physical layout of the FMC includes :

- Six modules of BOSCH flexible conveyor: This conveyor is used to transport a pallet with a working part to the work location and transport a finished part to the next workstation. Every module has proximity switches, pneumatic stop gates, pneumatic cylinders, motors, locator lifts and transfer lifts.
- A Puma 760 material handling robot: This robot with a six axis arm and parallel gripper can move raw material from a pallet into the vise of the machining center, and move a finished part onto a waiting pallet.
- A HAAS machining center which has a Reshiaw inspection probe: This HAAS VF-1 milling machining center with a tool changer of 20 tools capacity is used in this research to manufacture part and to measure a finished part for quality sampling. This research uses a Renishiaw probe to setup zero reference point for a workpiece before initiating the manufacturing process and measure its size after the process is finished. A workpiece's probing information is sent to the control computer via RS232 communication.

- A Pentium control computer. This Pentium computer performs process control for all the connected DCSs and an electrical vise, and it monitors and executes the real-time SPC activities.

The operation procedure consists of the following steps:

1. The conveyor transports a pallet with a workpiece to the loading and unloading location.
2. The PUMA robot pickups the workpiece and puts it into an electrical vise which is mounted on the machining center's work table.
3. The electrical vise closes and fixes the workpiece. The Puma robot's arm moves to the safety position.
4. The machining center starts to manufacturing the workpiece.
5. After the workpiece is finished, the vise opens and the robot moves the workpiece back to the waiting pallet.
6. The pallet leaves for the next station and the robot moves to the standby position to wait for next pallet's arrival.

If the control computer decides to do sampling for SPC after the workpiece is manufactured, the inspection probe will measure the desired dimension, for example, a diameter of a hole. The MC will send the probing information to the control computer. The control computer will plot $\overline{X}$ and R charts and implement other SPC activities.

4-3 Integration Architecture Implementation

The framework for the physical manufacturing system in Figure 4-1 is shown in Figure 4-2. This FMC's development is based on the integration architecture discussed in Chapter 3. This framework emphasizes, at a higher level, the integration activities and mutual relationship for DCSs. Some of the major components of Figure 4-2 are briefly explained below

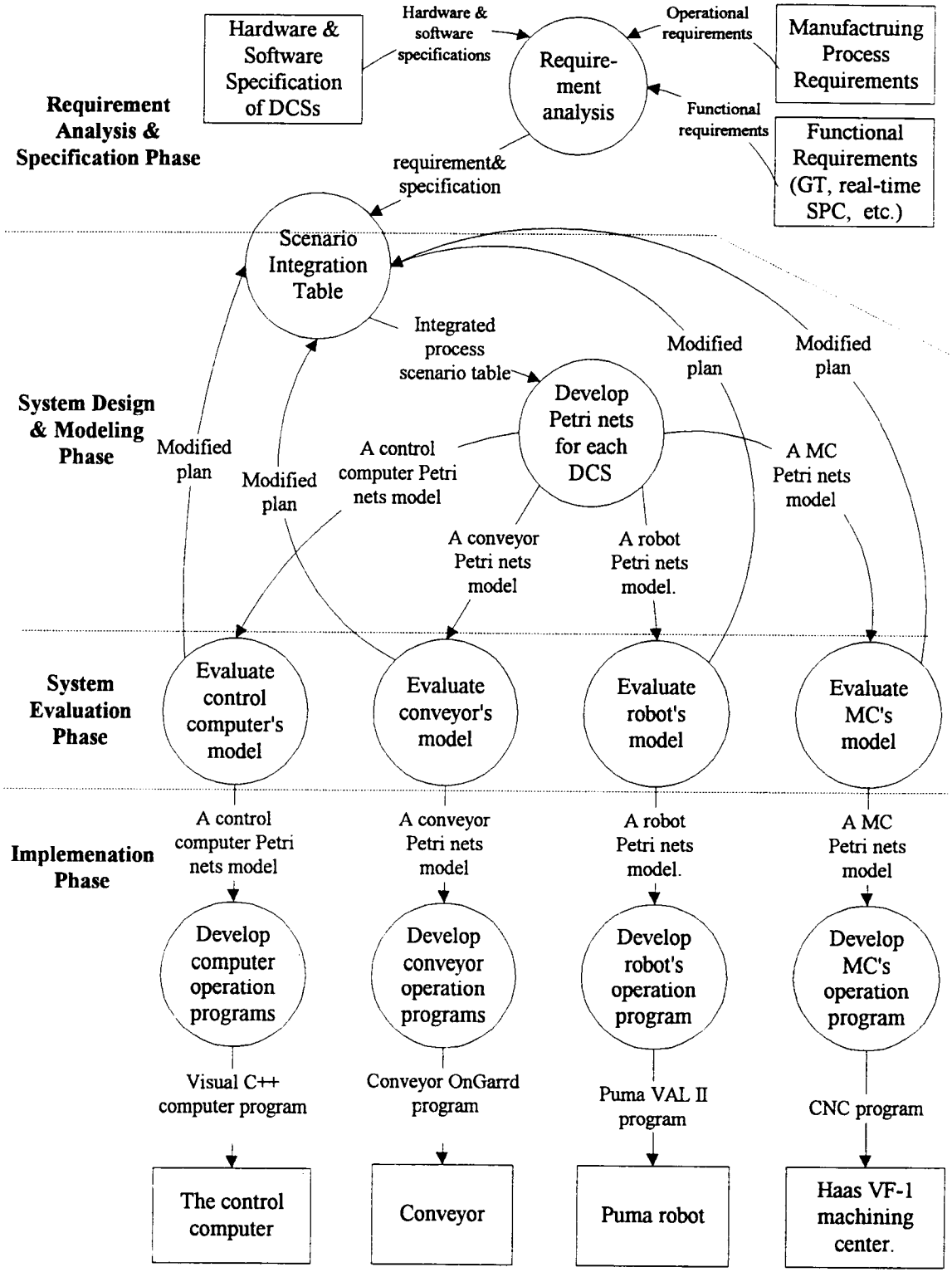


Figure 4-2 Implementation of the Integration Framework

4-3-1 Scenario Integration Table

The complete manufacturing scenario integration table for the system used in this research is given in Table 4-1.

Table 4-1 An Integrated Manufacturing Process Scenario

COMPUTER		CONVEYOR		ROBOT		MACHINING CENTER	
No	**Operation**	**No**	**Operation**	**No**	**Operation**	**No**	**Operation**
CC 1	Set up procedure & open the Vise in MC	C1	Set up procedure	R1	Set up procedure	M1	Set up procedure
CC 2	Polling the conveyor Send(Conveyor, Ready)	C2	Wait a "Ready" message from computer Wait(Conveyor, Ready)				
CC 3	Wait for a "Ready" message from the conveyor. Wait(Conveyor, Ready)	C3	Return a "Ready" message. Send(Conveyor, Ready)				
CC 4	Send an "Ack" to the conveyor. Send(Conveyor, Ack)	C4	Wait for an "Ack" Wait(Conveyor, Ack)				
CC 5	Polling the robot. Send(Robot,Ready)	C5	The loading location available	R2	Wait a "Ready" message Wait(Robot,Ready)		
CC 6	Wait for a "Ready" message from the robot Wait(Robot,Ready)	C6	Let the waiting pallet enter the loading location	R3	Return a "Ready" message. Send(Robot,Ready)		
CC 7	Send an "Ack" to the robot . Send(Robot,Ack)	C7	A pallet arrives at the loading location	R4	Wait for the "Ack" Wait(Robot, Ack)		
CC 8	Polling the Machining Center. Send a "Ready" Send(MC, Ready)			R5	Move to the standby position1	M2	Wait a "Ready" message Wait(MC, Ready)
CC 9	Wait for a "Ack" message from the M.C. Wait(MC, Ack)					M3	Return a "Ack" message Ack Send(MC, Ack)
CC 10	Wait for an "Arrival" message from the conveyor Wait(Conveyor, Arrival)	C8	Send an "Arrival" to the computer. Send(Conveyor, Arrival)			M4	To start a cycle Standby procedure
CC 11	Send an "Ack" to the conveyor Send(Conveyor,Ack)	C9	Wait for an "Ack" Wait(Conveyor,Ack)				
CC 12	Send an "Arrival" message to the robot Send(Robot, Arrival)			R6	Wait for an "Arrival" message Wait(Robot,Arrival)		
CC 13	Wait for an "Ack" message from the robot. Wait(Robot, Ack)			R7	Send an "Ack" message Send(Robot,Ack)		
				R8	Move the part into the vise		
CC 14	Waif for "InVise" message from the robot Wait(Robot,InVise)			R9	Send a "InVise" message Send(Robot,InVise)		
CC 15	Close the vise						
CC 16	Send an "Ack" to the robot Send(Robot,Ack)			R10	Wait for an Ack Wait(Robot, Ack)		
CC 17	Wait for a "Clear" message from the robot			R11	Move to the standby position2		

Table 4-1 An Integrated Manufacturing Process Scenario (Continued)

CC 17	Wait for a "Clear" message from the robot Wait(Robot,Clear)			R12	Send a "Clear" message Send(Robot,Clear)		
CC 18	Send an "Ack" back to the computer Send(Robot,Ack)			R13	Wait for an "Ack" Wait(Robot,Ack)		
CC 19	Make decision for manufacturing the part with or without probing			R14	Wait for the working part finished		
CC 20	Send a "Start/Probe" or "Start/No Probe" to the MC. Send(MC, Start/xx Probe)					M5	Wait for a start message Wait(MC,Start/xx Probe)
CC 21	Read and decode the RS232 message form M.C. , If receive an "Ack" go to step 23 Wait(MC, Ack)					M 6a	If the received message is valid message, then send an "Ack" and goto M7 Send(MC,Ack)
CC 22	, If receive an "Nack" go to step 20 Wait(MC,Nack)					M 6b	If the received message is invalid message, then send an "Nack", and goto M5. Send(MC,Nack)
						M7	Use the probe to detect the part zero reference point
						M8	Manufacture the part in the vise.
CC 23	Wait for RS232 message from the M.C. If the decision for the MC was "Start/ Probe".					M9	If the message is "Start/ Probe", then goto M12
CC 24	Wait for a "Finished" from MC Wait(MC,Finished)					M 10	send "Finished" and Send(MC,Finished)
CC 25	Send(MC, Ack) goto to 28					M 11	Wait(MC,Ack) then goto M15
						M 12	Probe the diameter of the part
CC 26	Receive the probing data. (the probe data processing is in step 31) Wait(MC,probedata)					M 13	Send the probed information Probing data Send(MC, probedata)
CC 27	Send(MC,Ack)					M 14	Wait(MC,Ack)
CC 28	Open the vise						
CC 29	Send "Finished" to the robot Send(Robot,Finished)			R15	Wait for a "Finished" Wait(Robot,Finished)		
CC 30	Wait for an "Ack" from the robot Wait(Robot,Ack)			R16	Send an "Ack" Send(Robot,Ack)		
CC 31	Make a decision for M.C. and plot and make control chart if necessary.			R17	Pick up the part and put it back into the waiting pallet		
				R18	Move to the standby position		
CC 32	Wait for a "InPallet" from the robot. Wait(Robot,InPallet)	C 10	Wait for instruction for next cycle	R19	Send a "InPallet" Send(Robot,InPallet)		
CC 33	Send an "Ack" Send(Robot,Ack)			R20	Wait for an "Ack" Wait(Robot,Ack)		
CC 34	Send a "Next" or "Last" to the conveyor. Send(Conveyor,Next) or Send(Conveyor, Last)	C 11	Wait(Conveyor,Next) or Wait(Conveyor, Last)				

Table 4-1 An Integrated Manufacturing Process Scenario (Continued)

CC 35	Wait for an "Ack" from the conveyor Wait(Conveyor,Ack)	C 12	Send an "Ack" Send(Robot, Ack)				
CC 36	Send the decision(Next or Last) to the robot Send(Robot,decision)			R21	Wait for a decision("Next" or "Last". Wait(Robot,decision)		
CC 37	Wait for an "Ack" Wait(Robot,Ack)	C 13	Next conveyor buffer of a workstation available	R22	Send an Ack Send(Robot, Ack)		
CC 38	Send a decision(Tool Change/Next, ToolChange/ End, End or Next) to the MC Send(MC,decision)	C 14	The pallet leaves for next station	R23	If decision is a "Next"; then goto R5	M 15	Wait for a valid decision message (Tool Change/ Next, ToolChange/End, End, or Next) Wait(MC,decision)
CC 39	Decode a RS232 message . If its an "Ack" then next step (40). Wait(MC,Ack)	C 15	Send a "Left"	R24	Ending Procedure	M 16a	If the received message is valid, send an "Ack" and goto M17 Send(MC,Ack)
CC 39	If the message is a "Nack" goto 38. Wait(MC,Nack)	C 15	Send a "Left"			M 16b	If the received message is invalid, send an "Nack" and goto step M15 Send(MC,Nack)
CC 40	Wait for "Left" message from the conveyor. Wait(Conveyor,Left)	C 15	Send a "Left" *Left* Send(Conveyor,Left)	R25	End	M 17	If the decision is need to replace the tool, then setup the tool changing flag.
CC 41	Send an "Ack" to the conveyor Send(Conveyor,Ack2)	C 16	Wait for an "Ack" Wait(Conveyor,Ack2)			M 18	If the decision is "Next" or "ToolChange/Next" goto M4
CC 42	If decision is Next; then goto CC10	C 17	If decision is Next; then goto C5			M 19	Ending Procedure
CC 43	Ending Procedure	C 18	Ending Procedure			M 20	End
CC 44	End	C 19	End				

Note:

1. Ack is a confirmation feedback for responding a sent message being received.
2. Like the Ack above, Ack2 is a confirmation signal as well. It is used to distinguish the Ack when two continue "Ack"s might cause system deadlocked.

4-3-2 The FMC's Manufacturing Process Models Using Petri Nets

A manufacturing process in a cell can be represented as either a condition circle or an active box according to its nature. Tokens represent the working part flow, the communicating message, or the current operation flow for each DCS. The control computer is the center of the FMC. It connects all DCSs and there is no direct linkage among DCSs. The complete integrated system (shown in Appendix-A) can be decomposed into: 1) the control computer and the conveyor system (Figure 4-3), 2) the control computer and the robot (Figure 4-4), and 3) the control computer and the machining center (Figure 4-5).

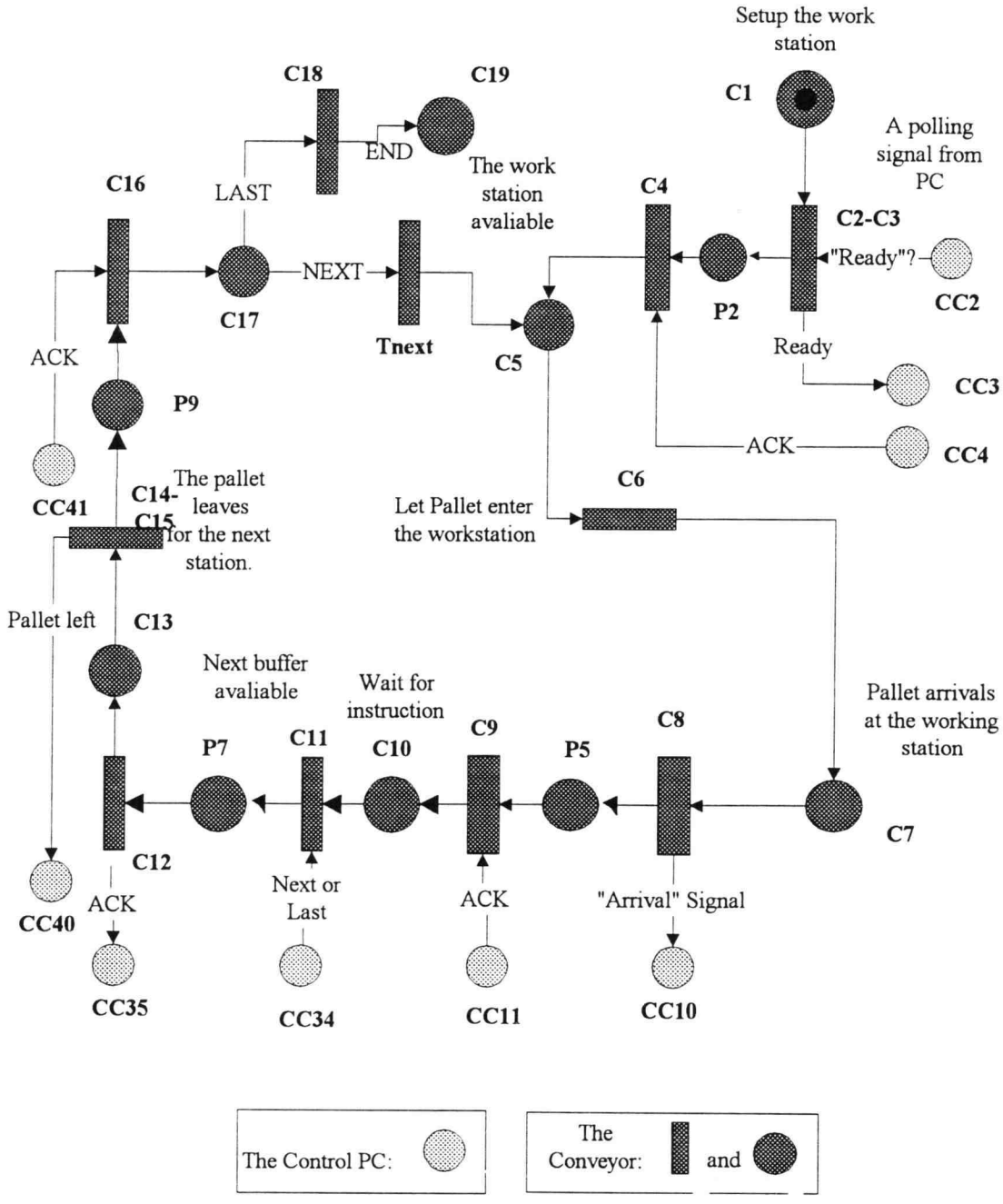


Figure 4-3 Petri Net Model for the Conveyor Control System

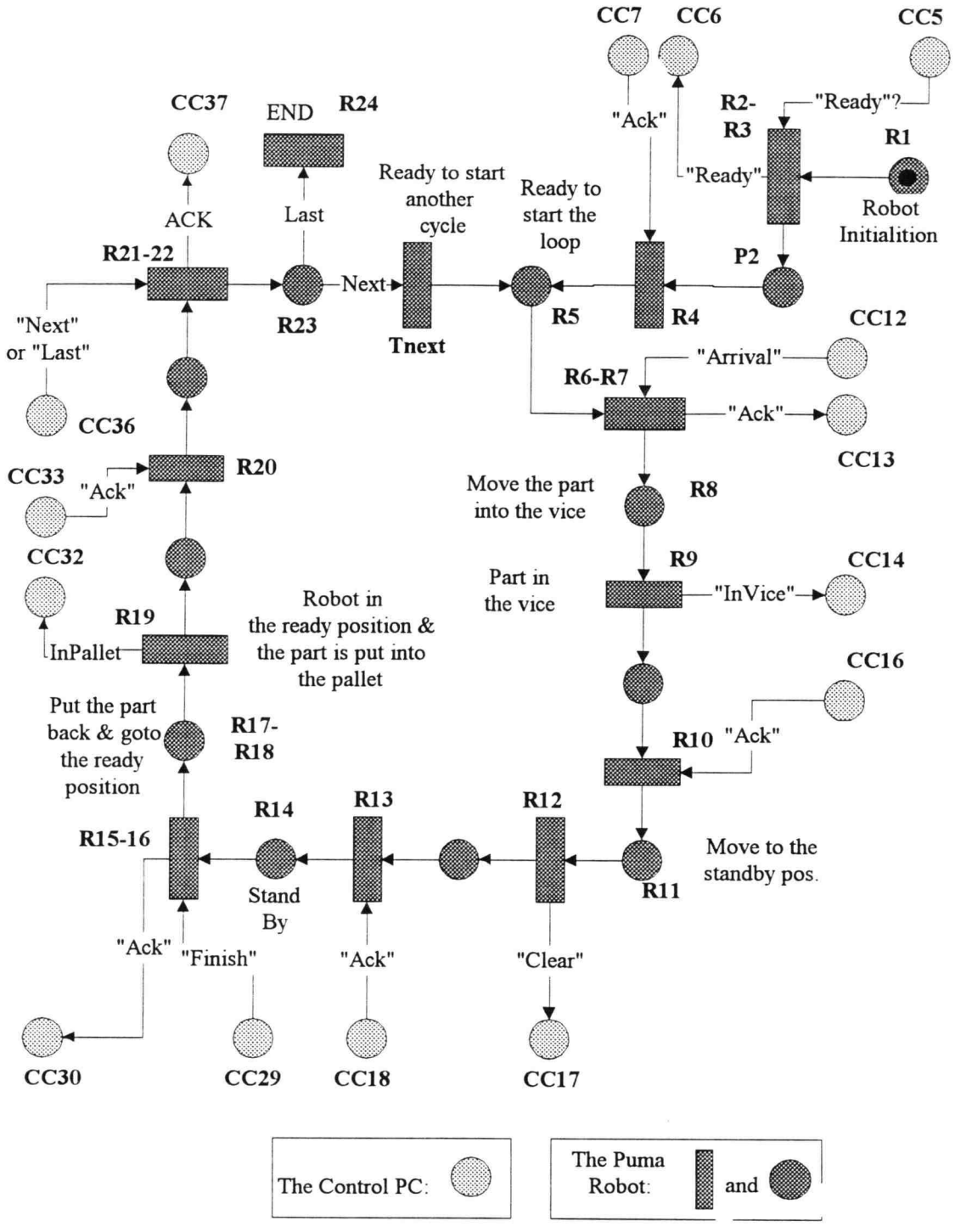


Figure 4-4 Petri Net Model for the Robot Control System

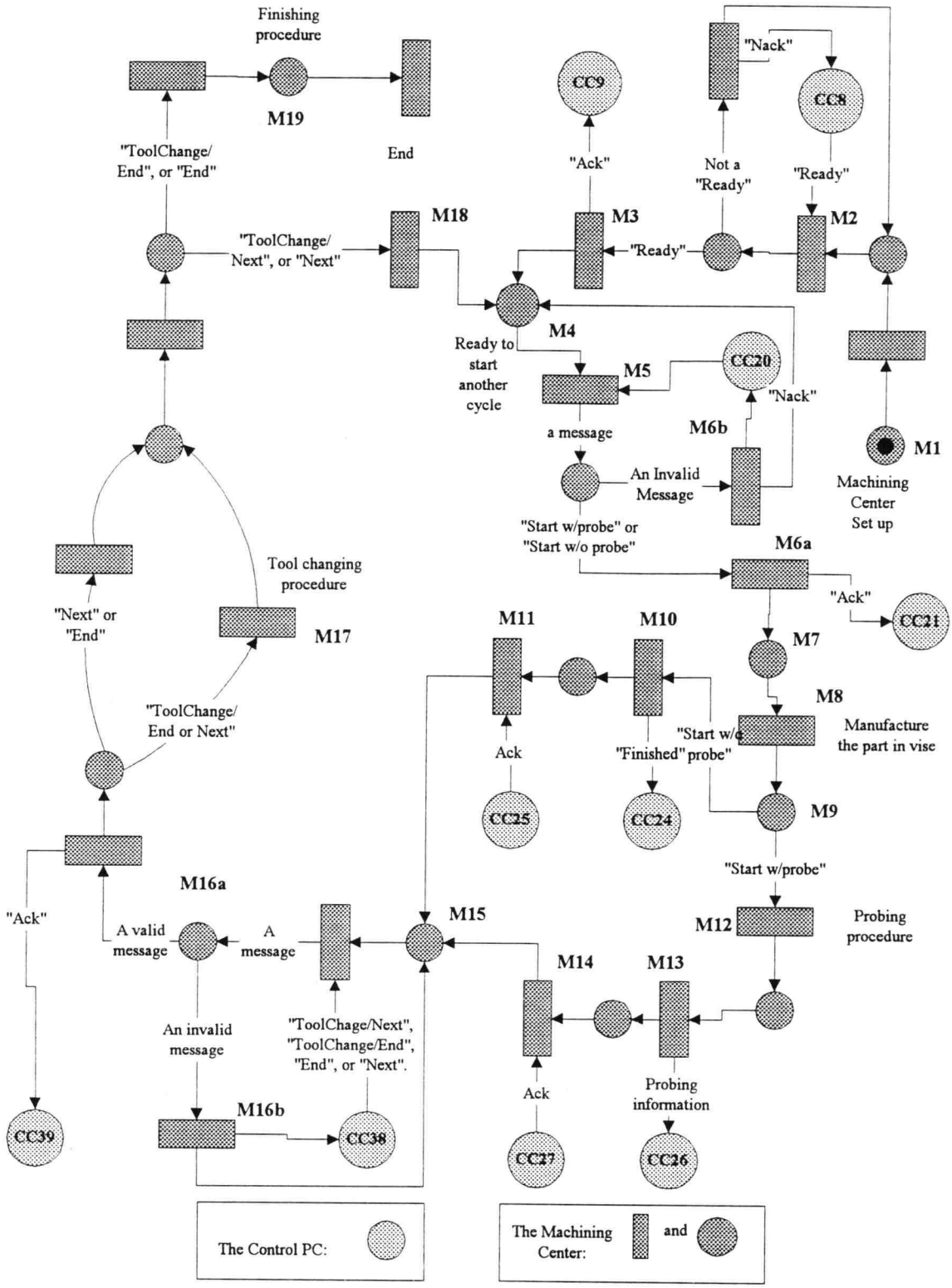


Figure 4-5 Petri Net Model for the Machining Center Control System

The procedure of translating a scenario integration table to a Petri net model, is explained with reference to the conveyor subsystem. If there is Send() or Wait() function in a cell of the Conveyor column in Table 4-1, the corresponding cell in the Control Computer column also needs to be considered. For example:

1. The cell C1, setup procedure, represents the beginning of conveyor Petri nets. Refer to Figure 4-3.
2. C2 and C3 can be combined together as a transition C2-3. If there is a Wait(station, command) procedure and is immediately followed by a return confirmation, Send(station, Ack), in next procedure , the two procedures can be combined as a transition. Here, C2 is a Wait(Conveyor, Ready) function and the responding Send(Conveyor, Ready) can be found at CC2 in the same row of C2 and under the Computer's column. CC3 can be found as the corresponding cell of C3 using C3 (Send(Conveyor, Ready)) to CC3 (Wait(Conveyor, Ready)). The C2-3 transition is enabled if its both input places have tokens. After this transition fires, it will send two tokens to both CC3 and place P2. This means that there are two concurrent processes for both the conveyor and the control computer. The input function of the transition C2-3 is I(C2-3)={C1, CC2}, and its output function is O(C2-3)={P2, CC3}.
3. P2 is a waiting place to temporarily store the token until the transition C4 receives an Ack from CC4. P2 is added into the Petri net to maintain the completeness of Petri net model and definition.
4. C4 is a transition where I(C4)={ P2, CC4} and O(C4)={ C5}. When the token is waiting in the place P2, the conveyor system is halted until an Ack from the control computer's CC4 is received. If both places have at least one token, the transition C4 will fire. After C4 fires, a token is sent to C5.
5. C5 is a start place of each cycle. Although C5 has two arcs from two different transitions, it can only receive a token from either C4 or Tnext at a time.
6. C6 is a transition where I(C6)={C5} and O(C6)={C7}. When C5's condition is fulfilled (the workstation available), C6 fires. This means that the conveyor will open the waiting buffer's stop gate to let the pallet enter the work location.
7. C7 is a place. In the conveyor system, the sensor in the work location will send a pallet's arrival signal to control computer (C8 fires) if a pallet arrives at the working location (C7's condition is

true). For Petri nets, the token will temporary stay in C7 until C7's condition is true (a pallet arrives at the work location) to enable the transition C8.

8. C8 is a transition where I(C8)={C7} and O(C8)={P5, CC10}. When condition of C7 is true, C8 fires with two tokens: one represents an "Arrival" signal to CC10, and the other token will be sent to place P5.
9. P5, like P2, is a place to temporarily store a token until a token from CC11 arrives.
10. C9 is a transition where I(C9)={P5, CC11}and O(C9)={C10}. When this transition fires, the token goes to C10.
11. C10 is a place. When a token goes into this place and a token arrives at CC11, the transition C11 fires. In the conveyor system , a pallet is waiting in the work location (C10) until the conveyor receives an instruction message (a token in CC34).
12. C11 is a transition where I(C11)={C10, CC34} and O(C11)={P7}. A token in CC34 represents a message, either a "Next" or "Last".
13. P7 is a place like P2.
14. C12 is a transition where I(C12)={P7} and O(C12)={C13, CC35}.
15. C13 is a place, the token stays here if the conveyor's next buffer is not available.
16. Like C2 and C3, C14 and C15 are combined together as the transition C14-15 where (C11)={C10, CC34}and O(C11)={P7}. When the workstation's next buffer is available (C13 is true), C14-15 fires and (a) the working station's stop gate opens to let the pallet leave for next station, (b) the token is sent to P9, and (c) a message, "Pallet left" (a token), is sent to CC40 to tell the control computer that the pallet has left.
17. P9 is a place similar to P2 .
18. C18 is a transition where I(C16)={C15, CC41} and O(C16)={C17}.
19. C17 is a place. It implements a conflict process for the transition Tnext and C19. When the place C17 has a token, it will enable either Tnext or C19 (but not both at the same time).
20. C18 is a transition where I(C18)={C17} and O(C18)={C19}. Tnext is a transition where I(Tnext)={C17} and O(Tnext)={C5}. Transitions C18 and Tnext are in conflict. Only one

transition can fire, since in firing , it removes the token in share input and disables another transition. If the decision from the control computer is a "Last", C18 fires, otherwise Tnext fires and a token goes to C5 starting another cycle.

21. C19 is a place which indicates the end of the conveyor process. If the token once goes into this place, it will stay there forever.

Evaluation of Petri Net Models

Evaluation using Petri nets includes:

- Check firing sequence: Compare the Petri net firing sequence and the Scenario Integration Table's operation.
- Check for reversibility: Check for reversibility of the initial marking from any reachable marking.
- Check for deadlock: Deadlock can be checked by identifying the deadlocked structure shown in Figure 4-6.

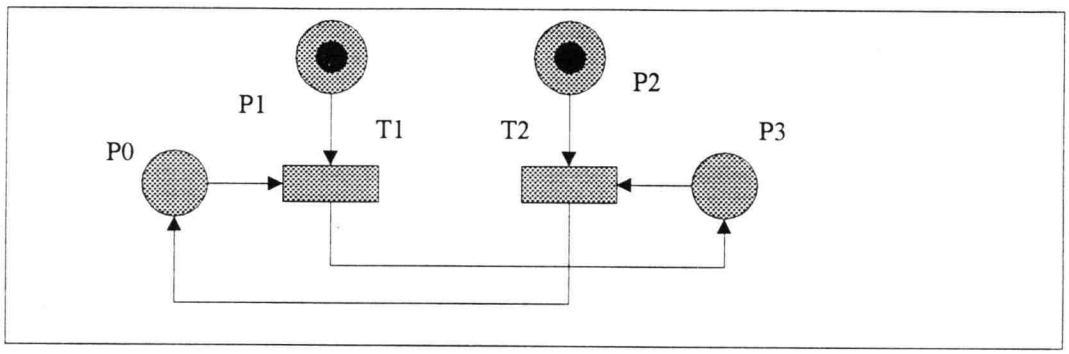


Figure 4-6 An Example of Deadlocked Petri Net

- Check for mutual exclusive condition: Check the required mutual exclusive implementation which is dealing with the impossibility of simultaneous submarking or firing concurrency, for example, C18 and Tnext in the Conveyor-Control Computer Petri net model in Figure 4-3. A mutual exclusive design for I/O Bus is given in Appendix D.

- Evaluate system utilization: Adjust the sequence of operational procedure to decrease DCSs' waiting time. This may involve deleting dead transitions (which can never be fired) and dead places (which can never be marked) or redefinition of some transitions.
- Check for boundness: Check for the characterizing fitness of the state space.

If a Petri net model is modified, then its scenario integration table must be updated as well. After the system testing phase, a final version of conveyor Petri net model is used to translate it into operational programs using appropriate language(s) with ease. The Petri nets models for robot and the machining center in Figures 4-3 and 4-4, respectively, follow logic similar to the conveyor model in Figure 4-5.

4-3-3 High Level Operational Programs for Manufacturing Process Implementation

In this stage, control computer's and conveyor's operational programs are developed, and robot's and MC's programs are developed as program templates. A program template for a DCS means a basic flexible program structure suitable for different kind of workparts. For example, a robot's program template, when applied to different part only need to update or fill in the part's pickup positions and place down positions using a teaching mode. To work a part in the machining center may only require input of the part's manufacturing and probing CNC calling function in the MC program template without the need to write new programs. This design can satisfy GT's requirement of the FMC.

Finally, for the control computer, many function calls are developed using Visual C++ to support users in translating the integrated procedure into source codes for the "procedure" function. For this research application, some consecutive operational procedures are grouped together because the application allows users to not only run it in continuos mode, but also in step mode. The users can monitor the DCS's operation status and interrupt an operation any time. Besides the FMC's integration execution, the control computer can also execute real-time SPC capabilities.

The flowcharts, which follow the final modified scenario integration table and Petri net models for each DCS, are given in Appendix B. The control computer's high level operational function, procedure(), is given in Appendix C.

4-4 System Control Architecture Description Using Data Flow Diagram

The overall system design, represented as Data Flow Diagrams (DFD), is shown in Figures 4-7 and 4-8. Figure 4-7 is a level 1 DFD to describe the system overall architecture at a conceptual level. There are five external entities of this system. They are:

1. Users: The users use the control computer system to test individual test interface, set up the production and sampling parameters, run manufacturing production, monitor the DCS's operations and perform real-time SPC.
2. Conveyor: This is a BOSCH FMS conveyor with OnGaard automation language system.
3. Puma robot: This is a Puma 760 material handling robot with VAL II system.
4. Machining Center: This is a HAAS VF-1 machining center for manufacturing work parts and doing the sampling measurement using the inspection probe.
5. Vise: This is an electrical vise which is controlled by the I/O Bus of the control computer to open and close. This vise is mounted on the Haas machining center's work table.

There are four main control systems -- the computer control system, the conveyor control system, the robot control system, and the machining center control system for each DCS. In addition, a database stores sampling information and out-of-control handling information. The data format design among the external entities, the control systems and the database are specified using data flow arcs shown in Figure 4-7.

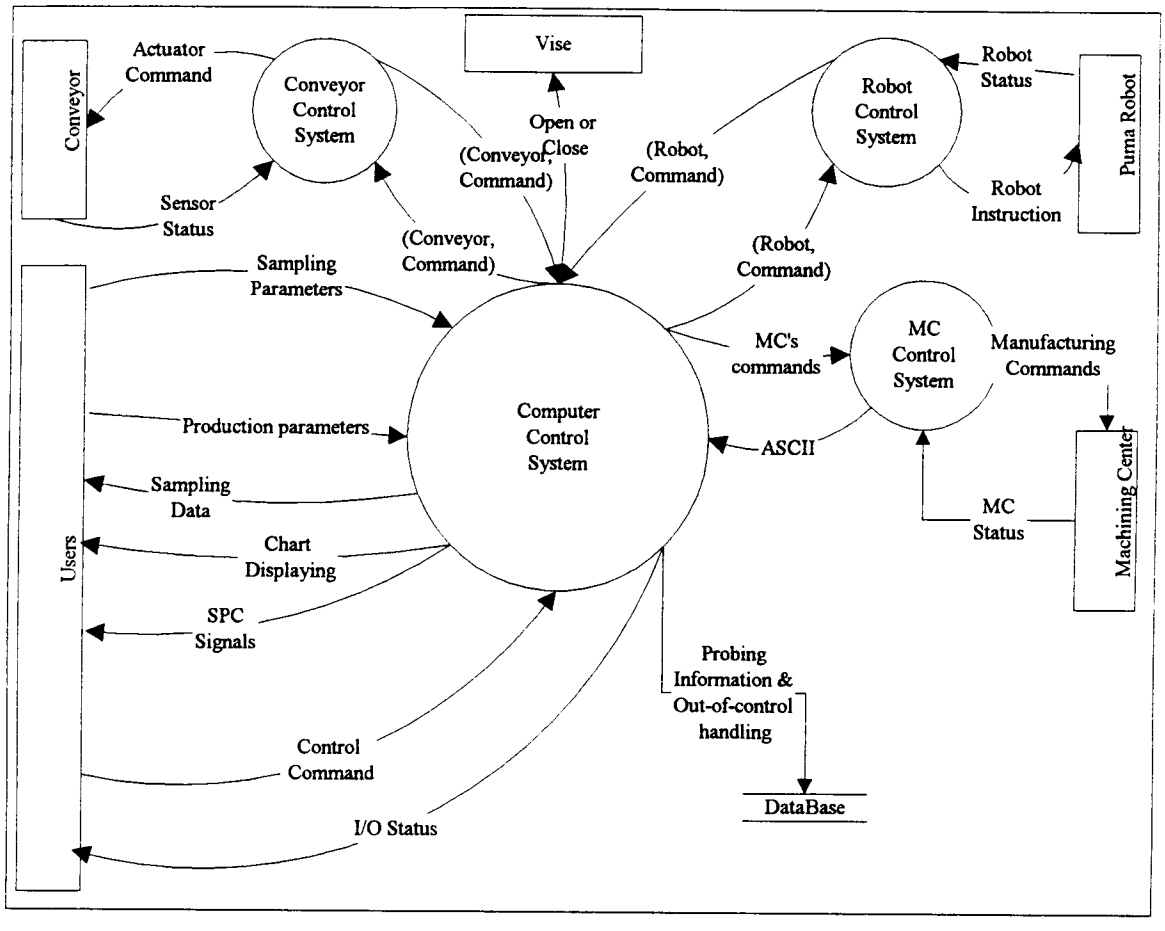


Figure 4-7 An Integrated System Data Flow Diagram (Level 1)

In the decomposition of Figure 4-7, the computer control system is decomposed into 1) set up procedure interface, 2) test I/O interface, 3) I/O bus status interface, 4) control chart interface, 5) integrated manufacturing procedure, 6) I/O bus protocol, and 7) RS232 decoding protocol. The conveyor control system is decomposed into conveyor operational program and conveyor's I/O bus protocol. Similarly, the robot and machining center (MC) control systems are decomposed into the robot and MC I/O bus protocols and robot and MC operational programs, respectively. These sub-processes are shown in Figure 4-8. Figure 4-8 shows level 2 DFD for further system design.

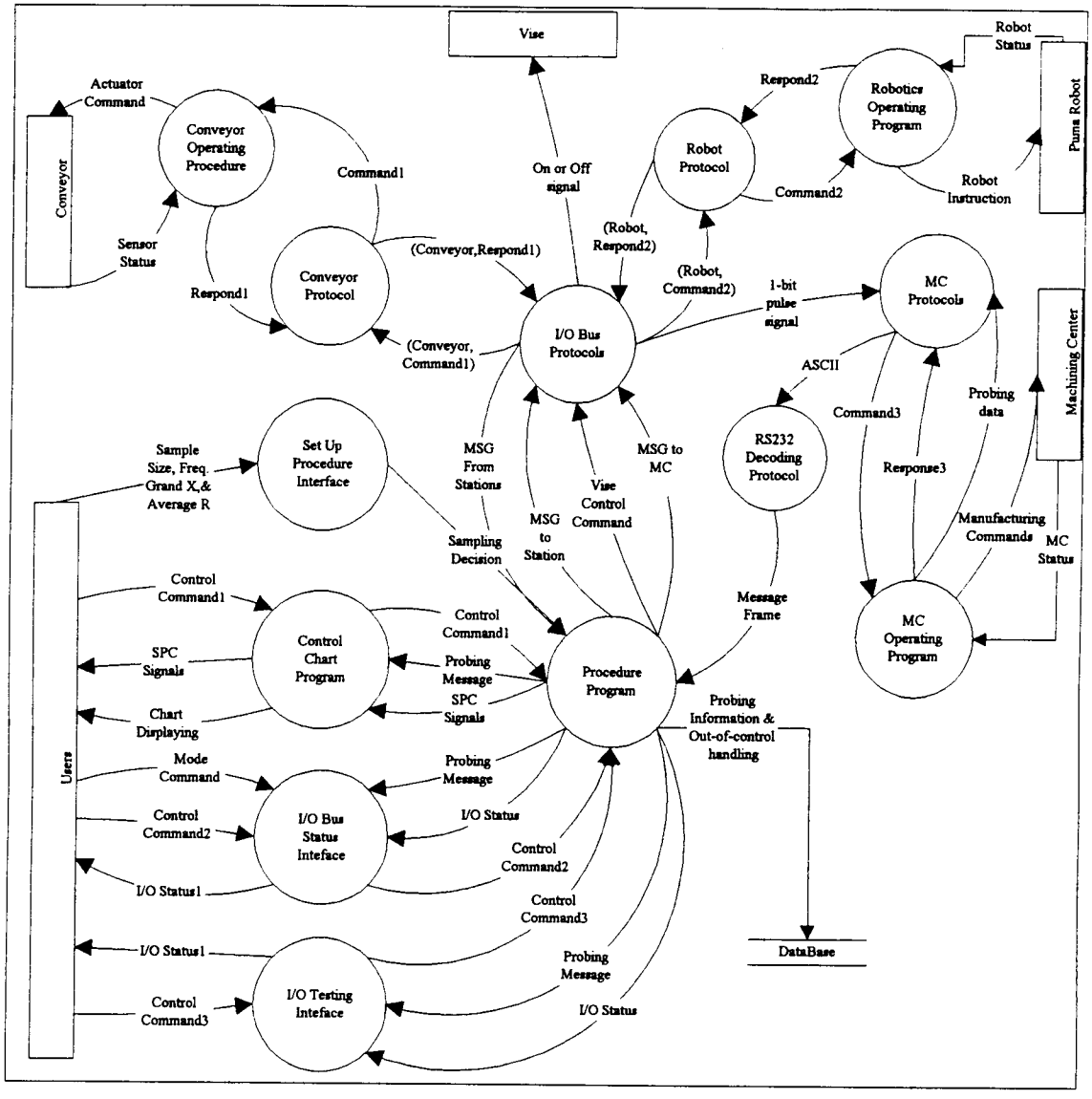


Figure 4-8 An Integrated System Data Flow Diagram (Level 2)

Graphic User Interfaces (GUI) are built in MS Windows environment. Users can use these interfaces to set up the communication linkage between the control computer and other components of the FMC, and to monitor the I/O signals for a manufacturing procedure. User can input parameters for each batch run. These parameters include: sample size, grand average ($\overline{\overline{X}}$), average of range ($\overline{R}$), sampling frequency, batch no. and production amount of a batch. During the manufacturing process, the sampling information and the two control charts, $\overline{X}$ and R chart, are presented to users. If there are out-

of-control signals, these SPC signals will be displayed to users. Users can also take the action that the control computer suggests in response to the out-of-control signal.

There are two communication interfaces, the "Test Interface" and the "Input/Output Status" in Figure 4-8 for communication construction and displaying I/O status in manufacturing procedures, respectively. The test interface has the capability to test each I/O function in communication. These functions involve sending a command to a desired destination, checking received status from connected machines, sending a signal to the machining center, checking RS232 received message and opening and closing the vise in the machining center. More detail about "Test Interface" is given in Appendix D.

The Input/Output Status interface is used to monitor a manufacturing process. A manufacturing procedure is developed using the scenario integration table and Petri nets as discussed in Section 4-3. A manufacturing process can not only be executed and finished within the end of a procedure cycle, but it can also be interrupted in any block. This interface shows the current I/O status and connected machines operation status during manufacturing process running. To fully monitor and set up the manufacturing process execution, this interface also offers the ability to run in a step mode. The step mode only runs one step at a time (its implementation is described in Section 5-4-2). This interface and the Test Interface were necessary during design and construction phase, and in tracing of errors.

At the start of executing a manufacturing procedure with a real-time SPC function, a user has to enter parameters into the "Sampling Set Up" interface. This "sampling sets up interface" calculates control limits, zone lines and center lines. This information, along with user input information, is sent to the procedure program for the SPC checking process.

In the "Control Chart" interface, the system can execute, end and interrupt the manufacturing process with real-time SPC checking. This interface offers users sampling data in a two-dimensional table from the machining center and $\overline{X}$ and R control chart displays. Besides the displaying functions, the

procedure program also performs SPC tests for the sampling data. Once an out-of-control signal is detected by a SPC test, an "Out-of-Control" warning window pops up to warn users and to let them decide whether to implement a remedy or not.

A relational database with two tables is designed to store sampling data and the out-of-control signal. For this purpose, a relation database with two tables is created. This database has two tables, "Sample" and "Out-of-control". The primary key (represented in Tables 4-2 and 4-3) for the table "Sample" is a concatenated key that consists of "batch no" and "sample no". The primary key for the table "Out-of-control" is also a concatenated key that consists of "batch no", "sample no" and "violated test". The database format is the following:

Table 4-2: Sample Data Table

*Batch No	*Sample No	Data1	Data2	Data3	Data4	$\overline{X}$	R	Date	Time
1	2	1.550	1.551	1.549	1.550	1.550	0.002	10/13/95	13:10

Table 4-3: Out-of-Control Records Table

*Batch No	*Sample No	*Violated Test	Chart Type	Action Log
1	23	Test X1:Extreme Point	$\overline{X}$	None
1	23	Test Y3:Linear Trend	R	Changed Tool

The table "Sample" is designed to record sample data in real-time. It can also be used in an off-line mode. The table "out-of-control" records all the out-of -control signals and the responding action, if any. The functional dependence structure for the database is shown in Figure 4-9.

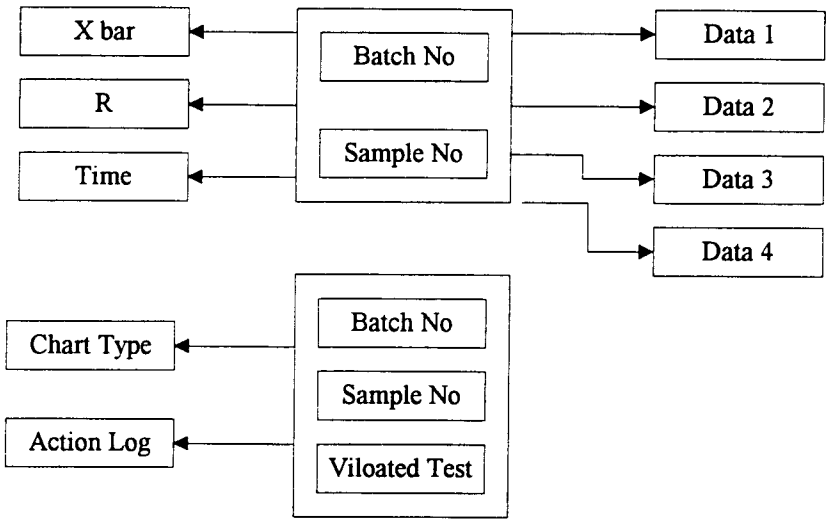


Figure 4-9 Functional Dependencies in the Sampling Database

To implement a manufacturing process, the conveyor, the robot, and the machining center must have their manufacturing process program. For example, the conveyor has an operating program to transport a pallet with raw material to a work location and to transport a finished part from a machining center to a next work location. The robot has an operating program to load the part from an arrival pallet on to the machining center and to unload a finished part from the machining center to a waiting pallet. Similarly, the machining center has an operating program to manufacture a part and to perform the probing procedure for sampling. A manufacturing process can be implemented using these operating programs and communication protocols.

4-5 Interfacing and Messages Passing Methods

A FMC can be defined as a node with its descendants in layer 4 of CIM architecture shown in Figure 2-2. The control computer is defined as a cell control node; therefore, it is the heart of the an integrated FMC and all other components are connected via the control computer.

Synchronization between the control computer and each DCS is implemented by message transmission. Safety in the manufacturing processes is very important for the operators and the machines. If a message error causes an improper procedure, it could harm the machines or the operators. Therefore, a confirmation feedback message ("Ack", an acknowledge signal) is implemented for each transmitted message. Another benefit for the connection-oriented services is that the sending bus can be released by the using DCS after it receives an "Ack" from the control computer.

The objective of the communication system design is to link all the DCSs in a cells. The communication system's range discussed in this research includes layer 4 and layer 5 (Figure 2-2). The fieldbus concept is applied in this communication system design. However, the BOSCH conveyor control system (OnGaard), the Puma robot control system (VAL-II) and the HAAS machining center system are not designed to support communication capability, such as MAP or Fieldbus protocols. Therefore, this research applies fieldbus's concepts in communication using low speed Solid State Relays.

The HAAS machining center is not designed to support interfacing with other machines. Consequently, a special design of the communication between the control computer and the MC has been developed to solve this problem. A general introduction of the communication methods used in this research is given below; For a more detailed description, see Appendix D.

4-5-1 I/O Bus Interface Architecture

The communication interface architecture among the control computer, the conveyor system and the robot is designed in three layers (Figure 4-10). The I/O bus applies the ISO/OSI reference model and fieldbus concepts and can support the following users: computer, robot, and conveyor. The HAAS MC can not apply this interface model because of its limitation. The application layer supports user's communication functions, Send(station, command) and Wait(Station, command) , to integrate DCSs. The Send and Wait functions are simple for connection-oriented services.

The I/O bus design for the FMC communication belongs to a broadcasting bus design (Figure 2-7 a). The input and output channels use 8 bits each. This I/O bus design has the advantages of flexibility, low cost, easy installation and safety. This design could be upgraded, if desired.

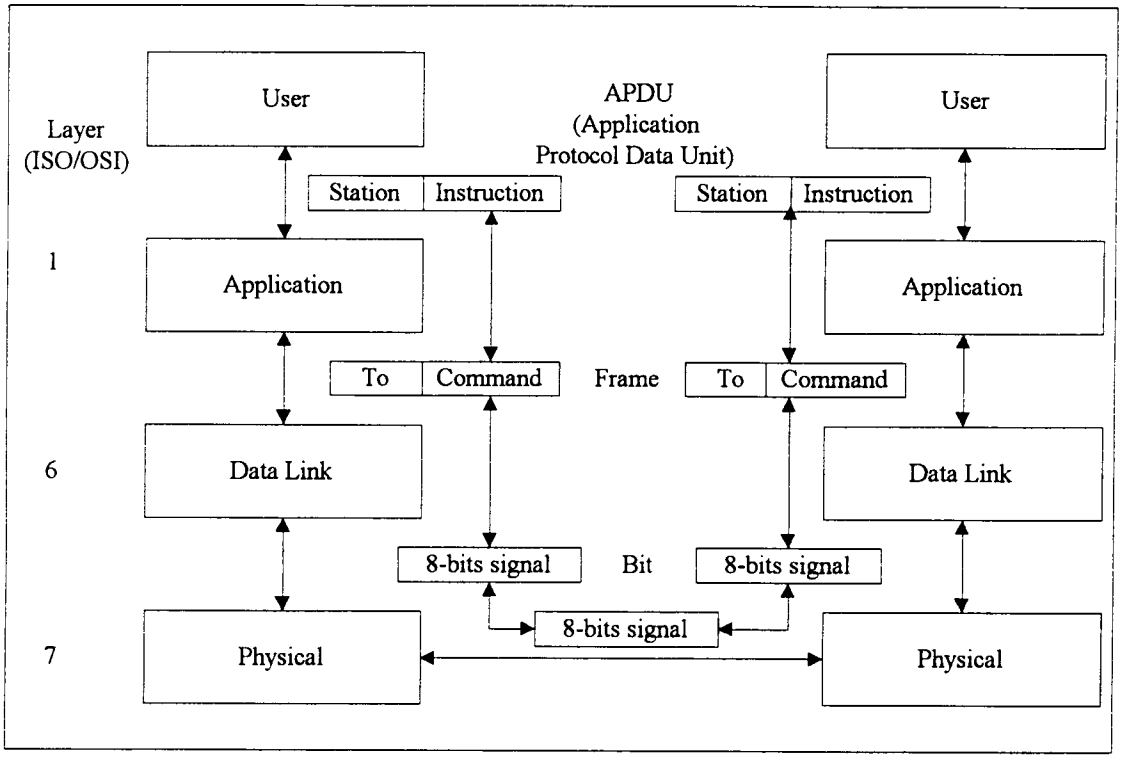


Figure 4-10 I/O Bus Architecture

4-5-2 The Communication Interface Architecture Between the Control Computer and the Machining Center

It was very difficult and time consuming to design the communication system for the HAAS MC because it only supports one bit input and 3 output relays. The machining center's timer and the one bit input port are used to receive a serial input pulse (like the old fashioned telegram) from the control computer and to decode the received signal to instructions. .

There is an unbalance structure (Figure 4-11) between the control computer and the machining center because HAAS machining center can not offer high level functions such as Send() and Wait() to

support the application layer. Also, there are two different transmitter mediums for the physical layer because HAAS machining center can not support I/O bus communication devise, it only has one input port, and it can send RS232 ASCII but can not receive RS232 ASCII code during run time. The communication structure of Figure 4-11 is necessitated due to HAAS's hardware (interface device) and software (the CNC code) limitations.

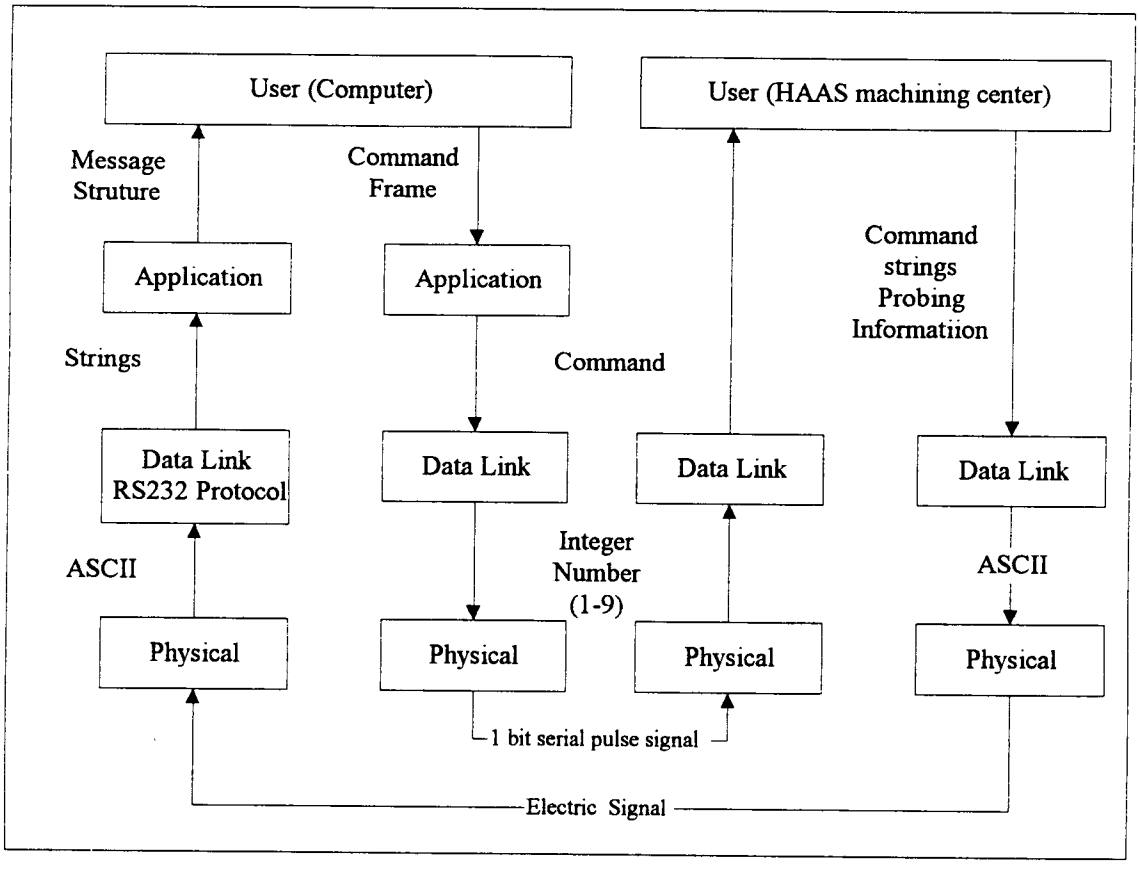


Figure 4-11 A Communication Interface Architecture Between the Control Computer and the HAAS Machining Center

CHAPTER 5 REAL-TIME SPC DESIGN AND IMPLEMENTATION

5-1 Overview

The $\overline{X}$ and R control charts are used for the FMC's real-time SPC requirement. The sampling and SPC checking procedure are designed for the FMC using both control charts. This SPC checking procedure design is described in Section 5-2. The $\overline{X}$ and R control charts' testing methods can be divided into three categories based on their characteristics. The real-time system needs very short response time; therefore, a fast algorithm is developed to detect SPC signals based on this three categories. Later, a SPC object-oriented model with embedded responding out-of-control recovery actions is developed using the three categories and the fast algorithm. These real-time SPC designs are discussed in Section 5-3. Finally, the FMC's integration and real-time SPC implementation and validation are described in Section 5-4.

5-2 Probing Information and Control Chart Displaying Design

When a current work part needs to be measured by the probe of the machining center (as decided by sampling frequency), the computer will wait for a probing signal. Originally, the received data from the machining center is a ASCII string for every transmit. The ASCII string is decoded and framed as message frames by the control computer using Windows RS232 protocol. The decoded frame is then translated into a float data for the work part dimension. More detail description is given in Appendix D.

This translated sampling data, a subsample data is then written into a grid table for display to users. If the collected subsample data are sufficient for sample data (the total subsample equals the sample size specified by the user in the sampling set-up dialog), the computer calculates this sample's average, $\overline{X}$, and range, R, and writes these values onto the grid table. The charting process function plots the $\overline{X}$ and R data into $\overline{X}$ and R graphs, respectively. The SPC test functions check the sample

data to detect any out-of-control signal. The sampling procedure and the real-time information display for $\overline{X}$, and R of each sample are shown in Figure 5-1.

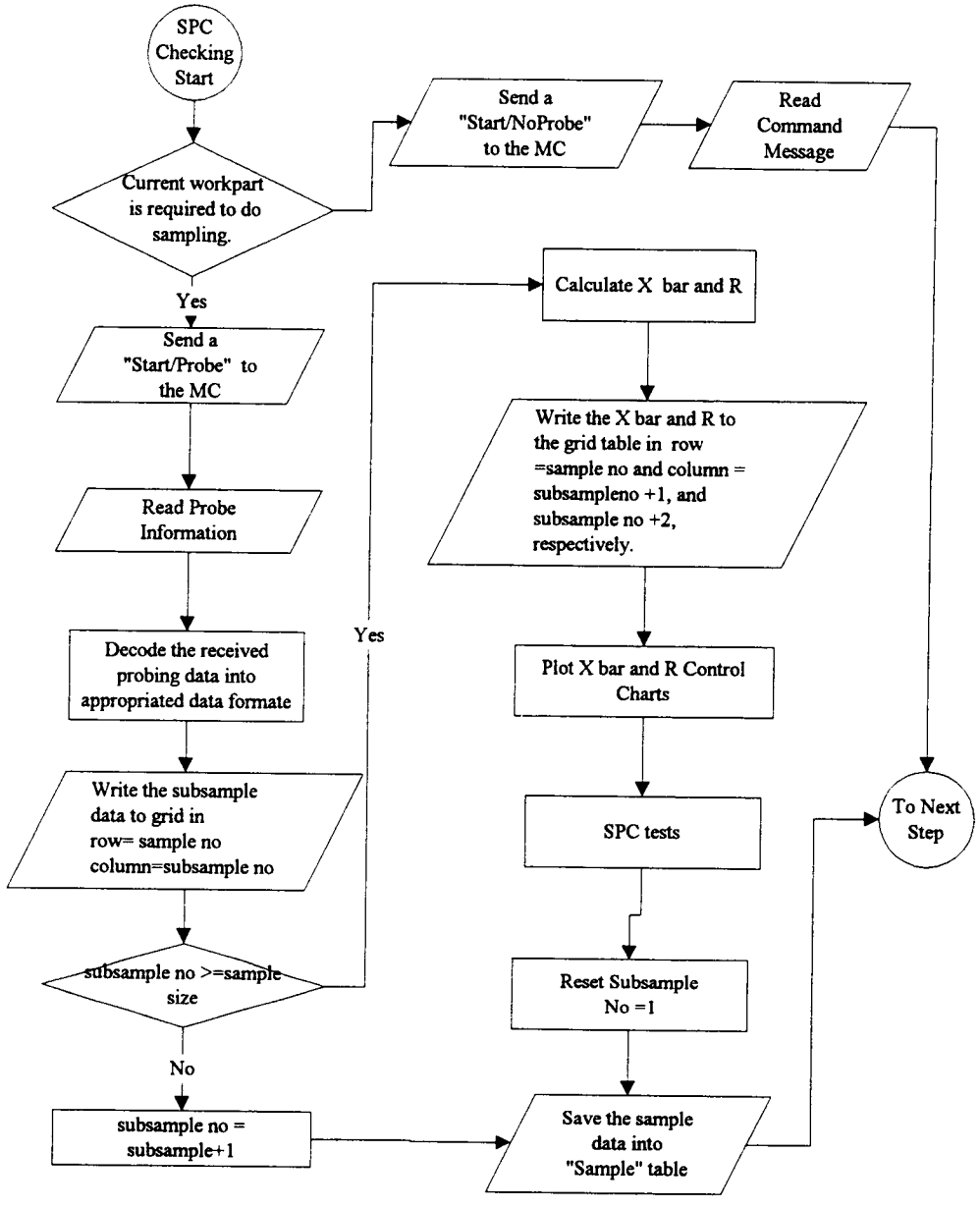


Figure 5-1 The Sampling and SPC Checking Procedure

5-3 Real-Time Statistical Process Control Design

The SPC for the machining center operation uses $\overline{X}$ and R charts to display the sampling status and Nelson's eight tests are used to detect an out-of-control signal. The software for each test is designed using object oriented concepts. Every test is designed as an object which inherits many attributes (data and functions) from the "Base Test" object. The Base Test object has basic information such as control limits, zone lines, center lines, etc. It has some functions which are used in derived objects such as: 1) SetUpLimits(): to get the sampling information, such as: $\overline{\overline{X}}$, $\overline{R}$ and sample size, which are set up by users in the sample dialog, and to calculate other parameters used in SPC tests, 2) WarnnDlg(): for handling the out-of-control dialog, and 3) Virtual functions SPCTests(): for SPC tests, Condition(): for checking the sampling data, and Action(): for out-of-control handling. The derived test objects have their own unique data members, functions and override functions SPCTests(), Condition() and Action(). This SPC object-oriented model is shown in Figure 5-2.

The eight tests for $\overline{X}$ and R charts can be divided into three different categories (Extreme points, X-Out-Of-T and Successive points in Table 5-1) based on their testing method.

Table 5-1 Modeling SPC Tests (for test numbers, refer to section 2-1-3-3)

CATEGORY	$\overline{X}$ TESTS	R TESTS
Extreme Point	1	1
X-Out-Of-T	2 and 3	
Successive Point	4, 5, 6, 7 and 8	4, 5 and 6

These three states can be modeled using finite state automata. The definitions are the following:

- State 0: The state 0 is a start state, an entering state. It is reached in the beginning of sampling.
- Current State: On starting a SPC test, a current state pointer points to the current state, Current State.

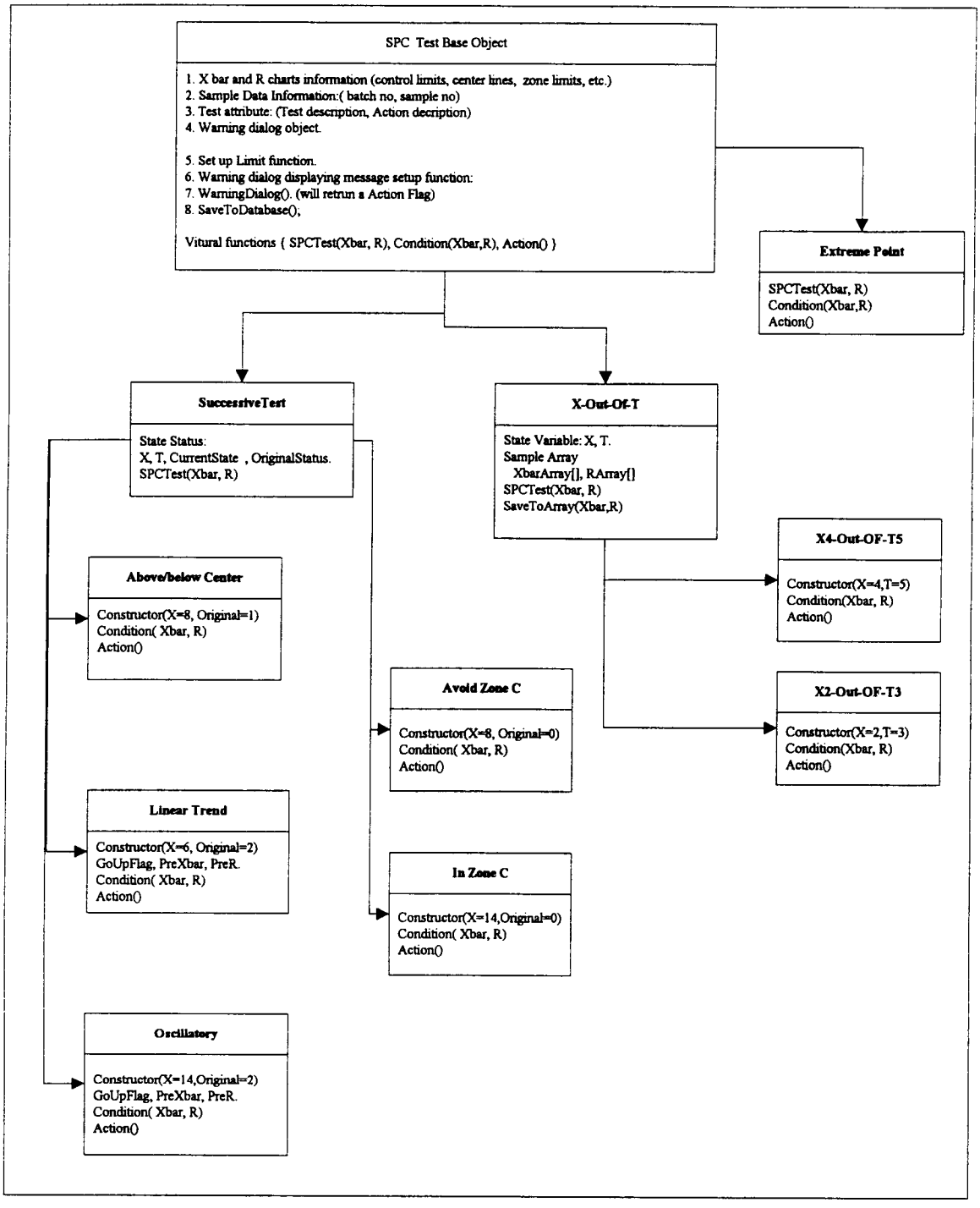


Figure 5-2 Object-Oriented Model for SPC Tests

- Condition function: The condition will analyze an input sampling data, the $\overline{X}$ and the R, return "TRUE" if it satisfies condition's functions, or "FALSE" if it does not. In the category two, X-Out-T, a Condition function returns "Forward", "Stay", and "Backward".
- X State : It is a trigger state which means that if the Current State points to this state or beyond, an out-of-control signal is generated.
- T State: It is a boundary state which is the maximum state number in a finite state model. The Current State can not go forward beyond this state.
- Original State: It is a restart point of a loop. Once the Current Sate enters this state, it is impossible to go backward.

5-3-1 Out of Control Signals Detecting

Category 1: Extreme Point:

In this category, both the Trigger state X and boundary state are one (shown in Figure 5-3). The state 0 is not only an entering point but also an Original State. This category can model Extreme point tests for the $\overline{X}$ and the R charts. The SPC Test function, for example, in Extreme points for $\overline{X}$ chart, takes the sampling data, then uses Condition functions to check whether this sample's $\overline{X}$ is out-of-control or not. An out-of-control signal would be generated if the ($\overline{X}$ > UCL) or ($\overline{X}$) < LCL. After the Current State points to State 1 and an out-of-control signal is generated, the Current State always points back to State 0.

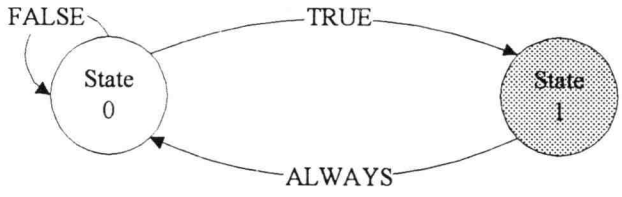


Figure 5-3 Extreme Point Category Finite State

The SPCTest algorithm for this category is as follows:

Procedure SPCTest(a collected sample data: $\overline{X}$ and R)

{ The input variable is the currently collected sample data}

Procedure begin

if the collected sample data satisfies the Condition() function test ***then***

begin

Display a warning dialog window ;

if user chooses to execute the responding out-of-control action ***then***

execute the function Action();

end;

Procedure end;

Category 2: X Out Of T Points

A SPC test in this category checks the current sample data and the previous (T-1) sample data to count the number of samples that satisfy SPC check condition. The finite state model can represent this category as shown in Figure 5-4. The condition checking for sample data will return three different results "Stay", "Forward", and Backward" for the state transition. This Condition function is defined as follows:

- The checking condition for sample data: For example, in test 2 for $\overline{X}$, the checking condition is "TRUE" if $\overline{X}$ for sampled data is in zone A or beyond; else "FALSE".
- The leaving state will happen when sample number is larger than the total number to be checked T. It is like a T cells queue (see Figure 5-5), the new received sample data will push the oldest sample data out of the queue.

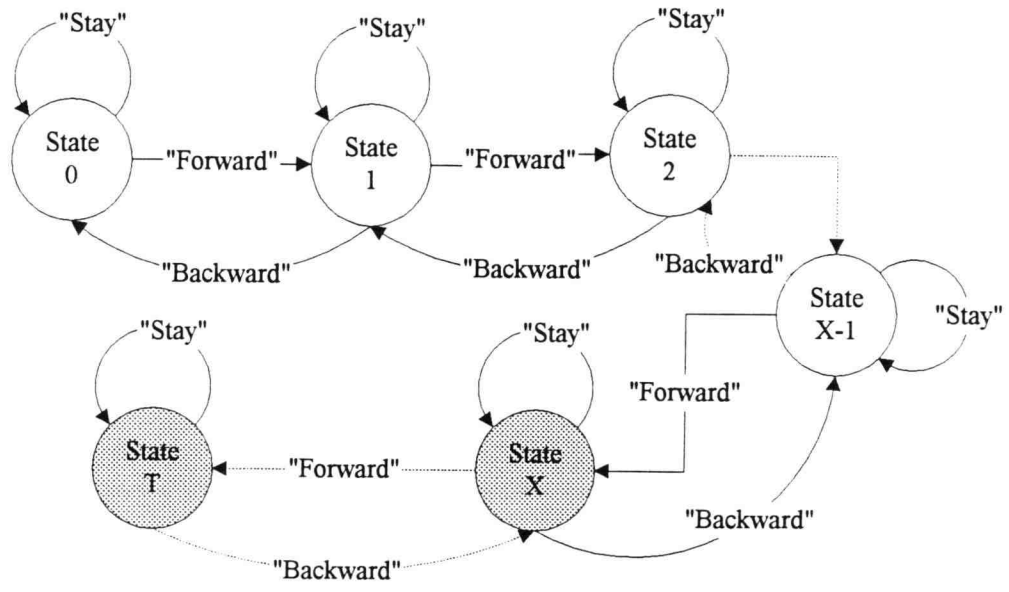


Figure 5-4 X-Out-Of-T Category Finite State

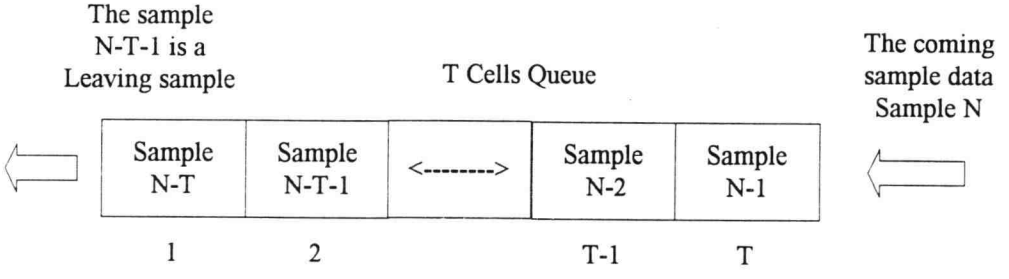


Figure 5-5 In-Coming and Leaving Sample Data

- Status A: The coming sample data satisfies the checking condition.
- Status B: The leaving sample data satisfies the checking condition.
- Status C: The Current State points to state zero.
- Condition returns "Forward" if ((A and C) or (A and ~B)) => (A and (~B or C)). The Current State points to next state.
- Condition returns "Stay" if ((~A and C) or (A and B)) => (A and (~B or C)). The Current State points stay the original state.

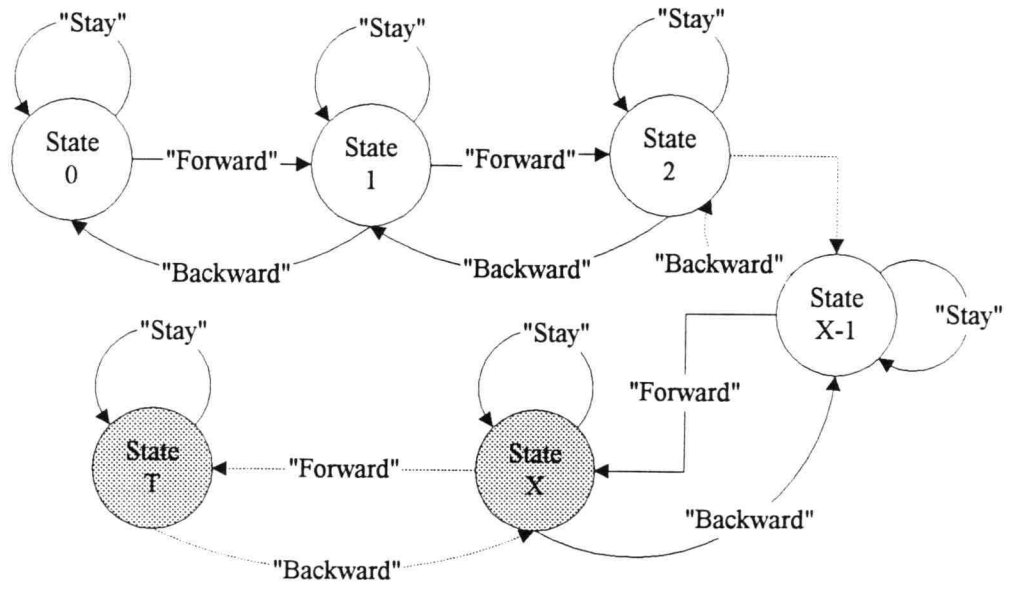


Figure 5-4 X-Out-Of-T Category Finite State

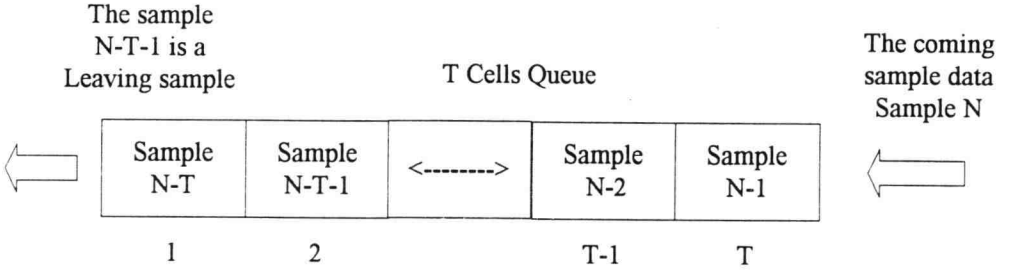


Figure 5-5 In-Coming and Leaving Sample Data

- Status A: The coming sample data satisfies the checking condition.
- Status B: The leaving sample data satisfies the checking condition.
- Status C: The Current State points to state zero.
- Condition returns "Forward" if ((A and C) or (A and ~B)) => (A and (~B or C)). The Current State points to next state.
- Condition returns "Stay" if ((~A and C) or (A and B)) => (A and (~B or C)). The Current State points stay the original state.

- Condition returns "Backward" if (~A and B)). The Current State points the previous state.

There is an alternative way to do this category of SPC test. Each object has an array T's sampling data. When an array is full , the oldest sampling data will be replaced by new arrival sampling data. During the SPC test, the array is checked and summed in order to count the satisfied sampling data's number by the test's Condition function. Finally, if the sum is larger or equal to the Trigger state, an out-of-control signal is generated. This approach is more straight forward than the approach based on Figure 5-4. Also, the condition function is consistent with the other two categories. The SPCTest function algorithm, based on this alternative approach, is given below.

Procedure SPCTest (a collected sample data: $\overline{X}$ and R)

{ The input variable is the currently collected sample data}

Procedure begin

Initialize the counter K=0;

Save the collected sample data into a T cells queue(see Figure 5-5) ;

{The following actions are counting the total number sample data which satisfy the Condition() function test in the T cells queue }

for CellNo=1 to T do

begin

if the collected sample data which is at the location CellNo of the T cells satisfies the Condition() function test ***then***

Increase the counter K by 1;

end;

if the counter K >= the trigger number X then

{ It is out-of-control}

begin

Display a warning dialog window ;

if *user chooses to execute the corresponding out-of-control action* ***then***

execute the function Action();

end;

Procedure end;

Category 3: Successive points:

The third category is the set of Successive point test (shown in Figure 5-6). In this category, the Current State number will be increased if the sampling data satisfies the Check Condition. Once a current sampling does not satisfy the test condition, the current state pointer will point back to the Original state.

- The $State_k$ will transit to $State_{k+1}$ if the in-coming sample data satisfies the checking condition, else the $State_k$ will transit to original status.
- A $State_k$ is out-of-control if $k >= X$, where X is the smallest number of successive sampling data which cause an out-of-control signal.
- When a $State_k$ is out-of-control , there will be an action to respond to it.
- The boundary T is less or equal (total desired amount)/ frequency.

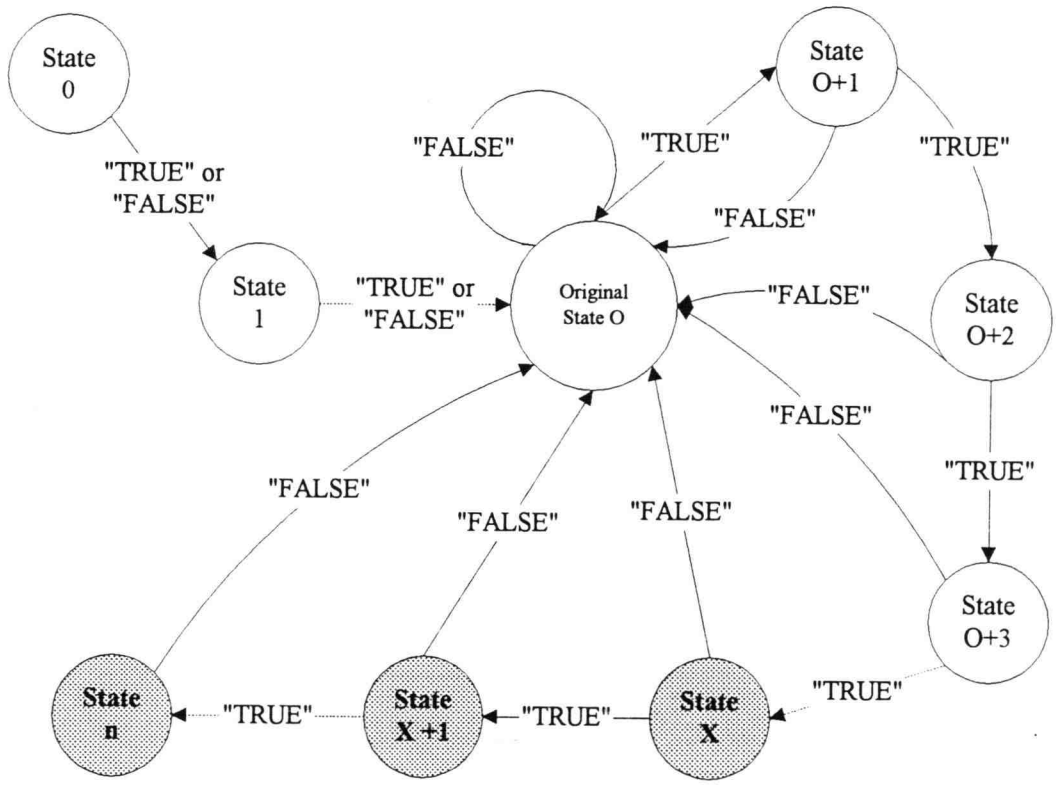


Figure 5-6 Successive points Category Finite State

The SPCTest function algorithm is as follows:

Procedure SPCTest(a collected sample data: $\overline{X}$ and R)

{ The input variable is the currently collected sample data}

Procedure begin

if *the collected sample data satisfies the Condition() function test* ***then***

begin

StateNo=StateNo+1; *// Current state pointer go forward one state.*

if *the current state pointer(StateNo) >= the trigger state X* ***then***

begin

Display a warning dialog window ;

if *user chooses to execute the responding out-of-control action* ***then***

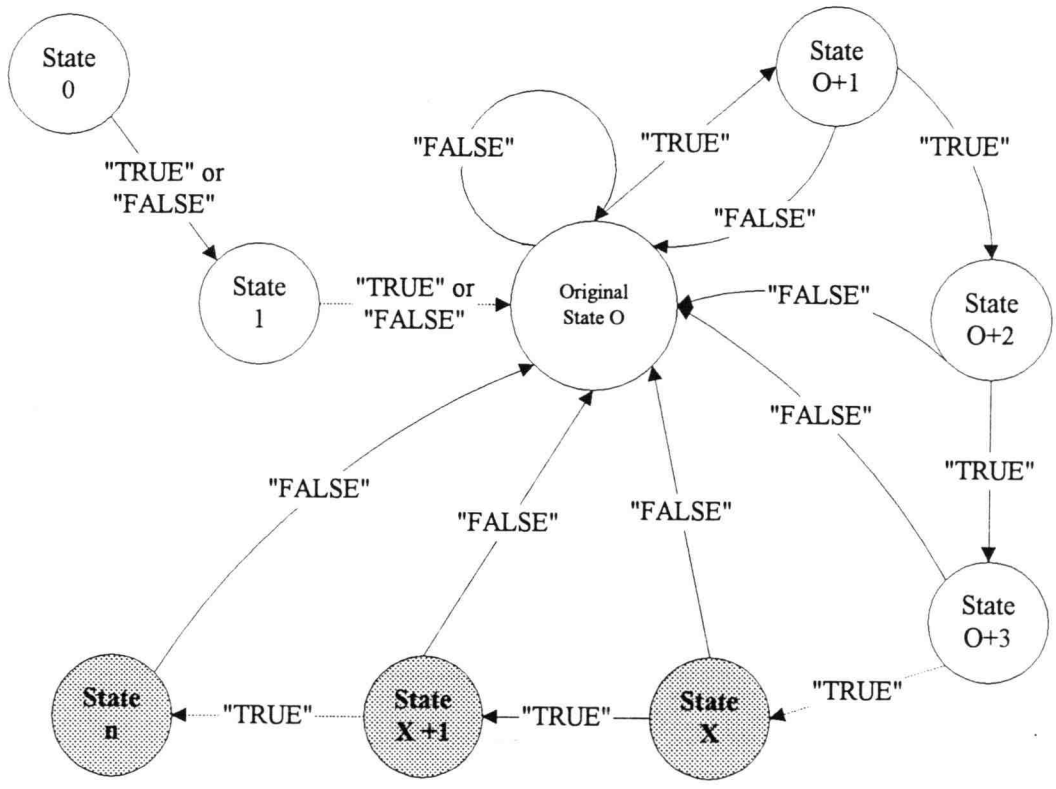


Figure 5-6 Successive points Category Finite State

The SPCTest function algorithm is as follows:

Procedure SPCTest(a collected sample data: $\overline{X}$ and R)

{ The input variable is the currently collected sample data}

Procedure begin

if *the collected sample data satisfies the Condition() function test* ***then***

begin

StateNo=StateNo+1; *// Current state pointer go forward one state.*

if *the current state pointer(StateNo) >= the trigger state X* ***then***

begin

Display a warning dialog window ;

if *user chooses to execute the responding out-of-control action* ***then***

execute the function Action();

else

Reset the current state pointer StateNo to the Original State;

end;

end;

Procedure end;

5-3-2 Out of Control Signals Handling

Every SPC test has an out-of-control handling procedure which is called by SPC test function of the SPC test. The handling function pops up a warning window with violated test description and a proposed action message. Users can either execute the suggested action or exit without any action. After the warning dialog is closed, the suggested action will be executed if users chose to implement the action. The appropriated out-of-control remedy action can be arranged into Action() function. For example, in this research when a downward linear trend and run below $\overline{X}$ chart's center line occurs in an $\overline{X}$ chart, a tool change command will be sent to the machining center, if this action is selected by the user. This action may be triggered by possible tool wear after a certain amount of processing (depending on tool life of the cutter, contacting time, and working part material). When the machining center receives a tool changing command, it will replace the worn out tool with another identical tool in the tool set of the machining center. These procedures are all implemented during run time. The system does not need to be stopped for tool set up.

5-4 Implementation and Validation

This section demonstrates the FMC's integration and real-time SPC capabilities. Before executing an integrated manufacturing operation (either using the "Input/Output Status" interface or the "Control Chart" interface), production parameters and sampling parameters must be defined (discussed in Section 5-4-1). The "Input/Output Status" interface is used to test the FMC's integrated manufacturing

operation (discussed in Section 5-4-2); The final FMC's integrated manufacturing operation can be run with the "Control Chart" interface(discussed in Section 5-4-3). Yet, it is almost impossible to fully test the FMC's real-time SPC capability in real manufacturing because of time and cost considerations. This chapter concludes with a discussion of a real-time simulator developed for this purpose.

5-4-1 Process Initialization

Before running the manufacturing process (either for I/O Status Test or SPC Control Chart execution), the production and sampling parameters must first be defined using the "SAMPLING SET UP" interface. This research assumes that an initial run for $\overline{X}$, $\overline{R}$, control limits and zone lines of $\overline{X}$ chart and R chart have already been completed based on a sample size of 4. The objective of the manufacturing process is to produce parts and to process real-time SPC's sampling procedure using the probe mounted in the machining center.

The production parameters that require specification are (Figure 5-7): 1) frequency, 2) production amount, and 3) batch number. They are related to manufacturing operations and discussed in section 5-4-1-1. The sampling parameters specified are: frequency, sample size, $\overline{X}$ and $\overline{R}$. These parameters are related to SPC activities and are discussed in Section 5-4-1-2.

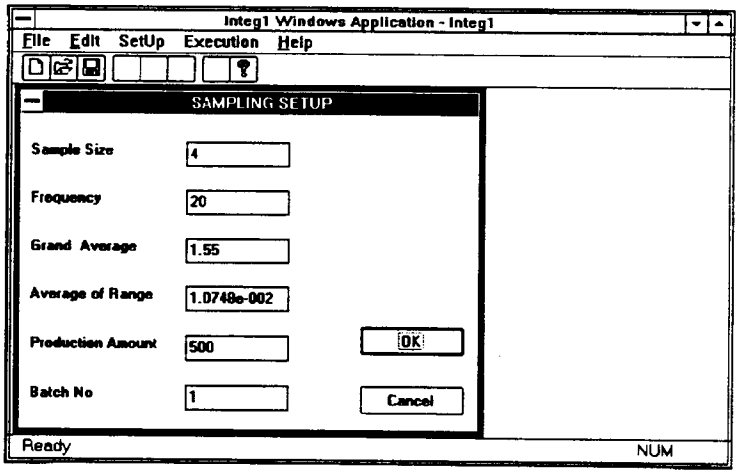


Figure 5-7 Production and Sampling Parameters Set Up Dialog

After the production and sampling parameters are input using the dialog box in Figure 5-7, the user clicks the "OK" button for execution. The use of the system is now illustrated using the part shown in Figure 5-8. The probe in the machining center measures a part's internal diameter.

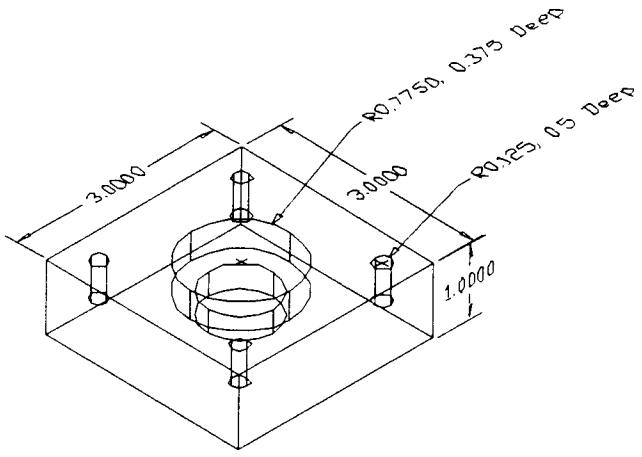


Figure 5-8 A Test Part's Drawing

5-4-1-1 Production Parameters

Besides providing an unique identification number for each sampled data point, the production parameters decide when the machining center needs to do sampling , how many parts need to be manufactured and when to terminate operation.

- Frequency: It decides when to do sampling. This research applies a distributed sampling method and assumes that after the machining center starts the manufacturing operation, the interval between successive part production time is constant. The sampling method takes a sample after a certain number of products are produced, the number determined by the frequency specification. For example, in Figure 5-7, the frequency 20 means that after manufacturing 20 parts, a sample is taken. Subsequently, samples will be taken after every 20 parts are produced. When a part needs to be sampled, the control computer will send a "Start/Probe" signal to the machining center. The machining center will use the probe to detect the part's diameter immediately after the part is finished and sends the probing results to the control computer.

- Production Amount: The "Production Amount" indicates the total desired number of parts for a batch production. The specification in Figure 5-7 indicates that after the machining center manufactures 500 parts, the control computer will send "Last" signals to the conveyor, the robot and the machining center to terminate the integrated manufacturing operation.
- Batch No: The "Batch No" with "Sample No" offers an unique key for every sample's record in the database as shown in Figure 4-9.

5-4-1-2 Sampling Parameters

- Frequency: As discussed in Section 5-4-1-1, frequency is both a production and sampling parameter.
- Sample size: Sample size is the number of parts in a sampling subgroup. This value in Figure 5-7 is 4 meaning that four consecutive samples will be grouped together to calculate the $\overline{X}$ and R using Equations 2.1 and 2.2. The calculated $\overline{X}$ and R value of a subgroup is shown in both the grid table and the control charts (Figure 5-10). The sample size also decides the A_2, D_3 and D_4 value for Equations 2.5 and 2.6.
- Grand Average: Based on the assumption discussed above, the Grand Average's ($\overline{\overline{X}}$) value is derived from an initial run for the part in Figure 5-8. This $\overline{\overline{X}}$ is the center line for the $\overline{X}$ chart.
- Average of Range: Similar to Grand Average, the average of range is based on an initial run. This $\overline{R}$ is the center line of the R chart.

Once the sampling parameters are set up, the center lines for both charts are defined. The control limits are also calculated using Equations 2.5 and 2.6 along with the zone limits (shown in Figure 2-8).

5-4-2 Manufacturing Process's Testing, Execution and Monitoring

After operational programs for each DCS have been developed to implement the integrated manufacturing process and individual DCS's communication functions have been tested using the test

interface discussed in Appendix D, an Input/Output Status Test interface (Figure 5-9) is used to test the integrated manufacturing operation for the FMC. To achieve this objective, this interface can run the entire operation for the FMC either in the normal continuous mode or in a step by step manner. This interface can also monitor the computer's input/output status, and the robot's, the conveyer's and machining center's operation status on the control computer's screen.

Before running this interface, the user must set up the production parameters. The production parameters (amount and frequency) are shown in the top-right corner of Figure 5-9; these production parameters are defined in the "SAMPLING SET UP" interface. This interface also displays the current product number and block number. For example, Figure 5-9 shows the FMC is processing the first part and the current working procedure is at block 12. The block is in row CC31 in the scenario integrated table (Table 4-1). The manufacturing process can be checked by comparing Table 4-1, this interface's display and the actual DCS operation.

To illustrate, Table 5-2 shows the manufacturing process in rows CC28, CC29 and CC30. This means that the following should be displayed in the interface: the vise opened, the control computer has sent a "Finished" signal to the robot, and the robot returned an "Ack" to the control computer. These processes status are displayed on the interface in Figure 5-9. Finally, a check of the corresponding DCS's operations will show that the conveyor is waiting for the finished part being put back into the waiting pallet, the robot is moving the finished part from the machining center to the waiting pallet on the conveyor and the machining center has finished manufacturing the part.

Table 5-2 Block Table for Mapping Table 4-1

Block	Computer	Conveyor	Robot	M.C.	Block	Computer	Conveyor	Robot	M.C.
1	CC1	C1	R1	M1	9	CC23			M9
1	CC2	C2			9	CC24			M10
1	CC3	C3			9	CC25			M11
1	CC4	C4			9				M12
2	CC5	C5	R2		9	CC26			M13
2	CC6	C6	R3		9	CC27			M14
2	CC7	C7	R4		10	CC28			
3	CC8		R5	M2	11	CC29		R15	
3	CC9			M3	11	CC30		R16	
4	CC10	C8		M4	12	CC31		R17	
4	CC11	C9			12			R18	
5	CC12		R6		13	CC32	C10	R19	
5	CC13		R7		13	CC33		R20	
5			R8		14	CC34	C11		
5	CC14		R9		14	CC35	C12		
6	CC15				15	CC36		R21	
7	CC16		R10		15	CC37	C13	R22	
7	CC17		R11		16	CC38	C14	R23	M15
7	CC17		R12		16	CC39	C16	R24	M16a
7	CC18		R13		16	CC39	C16		M16b
8	CC19		R14		17	CC40	C16	R25	M17
9	CC20			M5	17	CC41	C17		M18
9	CC21			M 6a	17	CC42	C18		M19
9	CC22			M 6b		CC43	C19		M20
9				M7		CC44	C20		
9				M8					

When an user chooses the "Step mode", the normal mode operation buttons are disabled (the user is not able to click the "Normal", "Run", "End" and "Exit" buttons). The user clicks the "Next Step" to execute the next block. The operation will be terminated when the current production number is equal to "Production Amount" or the user presses the "Stop" button. A manufacturing process must be tested in the step mode before it is used in the normal mode.

When the user chooses the "Normal Mode" and clicks the "Run" button, the manufacturing process is executed at normal pace. All the buttons excepts the "End" and "Stop" button are disabled. The operation is stopped when the current production number is equal to the production amount, or the user clicks the "End" or "Stop". The difference between "End" and "Stop" is that the "End" option stops

all the operations at the end of the cycle, and the "Stop" option stops all operations at the end of the current block.

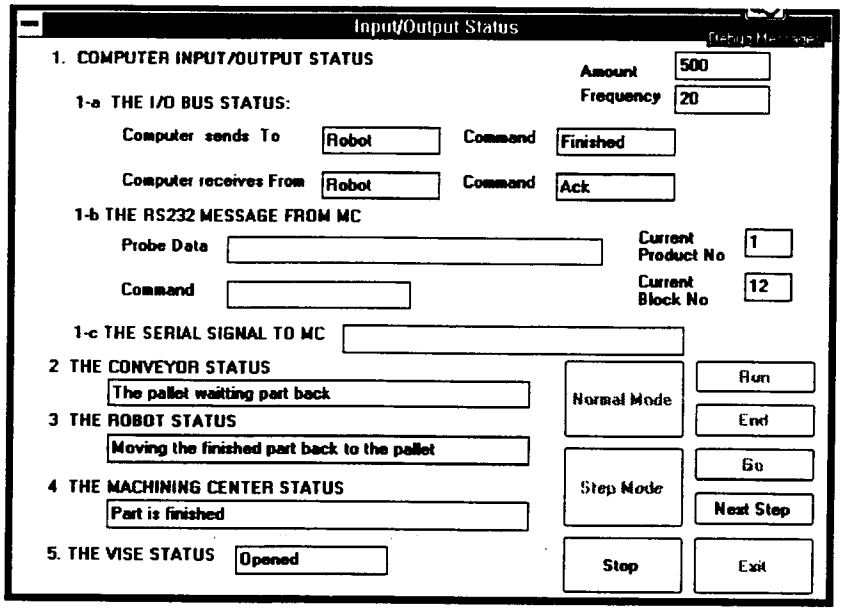


Figure 5-9 I/O Status Test Interface for Step Mode Execution

5-4-3 Sampling Data and SPC Control Charts Displaying

The control chart interface is used to execute an integrated manufacturing operation with real-time SPC procedure. When a part is sampled in the machining center, the probing data measurements are sent to the control computer. The control computer still needs to calculate the sampling because the machining center only sends offset, nominal and deviation ASCII strings to the control computer. The calculated sample data is stored in a two dimensional table as shown in Figure 5-10. Once a subgroup is constructed, its $\overline{X}$ and R are calculated and added to the grid table and plotted as a new point in $\overline{X}$ and the R charts.

The user clicks the "run" to start an operation. This interface can be treated as a "Normal Mode" of the "Input/Output Status Test Interface" using the "Run" button to start, the "Stop" or "End" button to stop the operation. This interface not only executes a manufacturing operation and displays its

sample data, but also implements the real-time SPC for the FMC process as discussed in the next section. In addition, this interface can printout $\overline{X}$ or R charts to a connected printer. Before running this interface, the production and sampling parameters must be set up in the "SAMPLING SETUP" interface.

To illustrate, Figure 5-10 shows that the currently received data is the first subsample of the 16th subgroup (sample no= 16, subsample no =1). This and prior data showed that 15 subgroups sampling data were received and their $\overline{X}$ and R are stored in the two dimensional table. On the right hand side, the $\overline{X}$ chart and R chart display the SPC information. There is no SPC signal in the process (as shown in Figure 5-10); the process is in control.

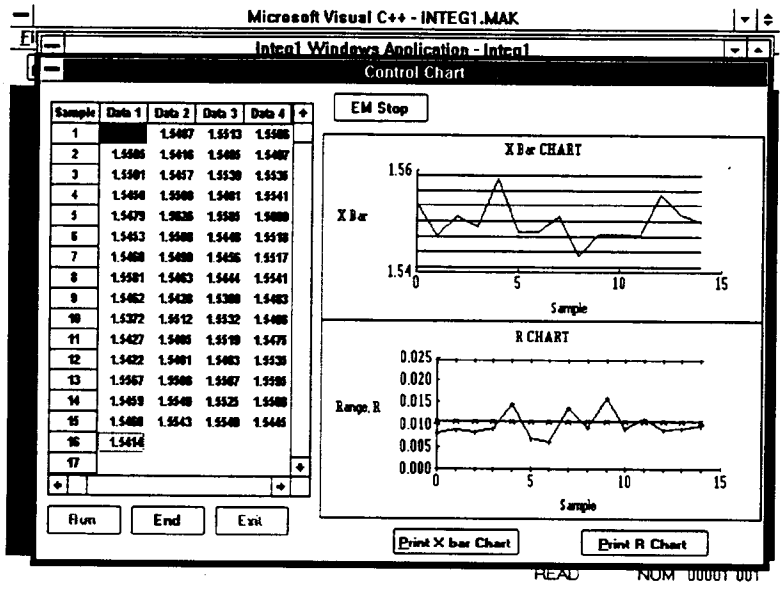


Figure 5-10 An In-Control SPC Control Chart Display

5-4-4 Real-Time SPC Implementation

For a real-time SPC application, it is very important to have a fast algorithm for SPC testing and the capability to resolve an out-of-control process problem immediately. The real-time SPC algorithm and its application in this research satisfies both of there requirements

5-4-4-1 Out Of Control Signals Detection Implementation

When the sampling data violates any of the tests for $\overline{X}$ or R charts, this out-of-control condition is detected using the SPC object-oriented model discussed in Section 3-5. For example, Figure 5-11 shows a FMC process that has been found to be out out-of-control because the control computer detects that the fifth subgroup sample data $\overline{X}$ is above the upper control limit (the test 1 of $\overline{X}$ chart). This means that a the process's mean may have shifted. In response to this out-of-control situation, an "Out-Of-Control Warning" dialog pops out with appropriate messages. The SPC test objects with extra condition checking can be used to detect a special cause such as a tool wear. For example, a "Tool Worn Out" is signaled in Figure 5-12 if the process is out-of-control indicated by downward linear. The linear trend test and run below/above test objects are used to test this cause.

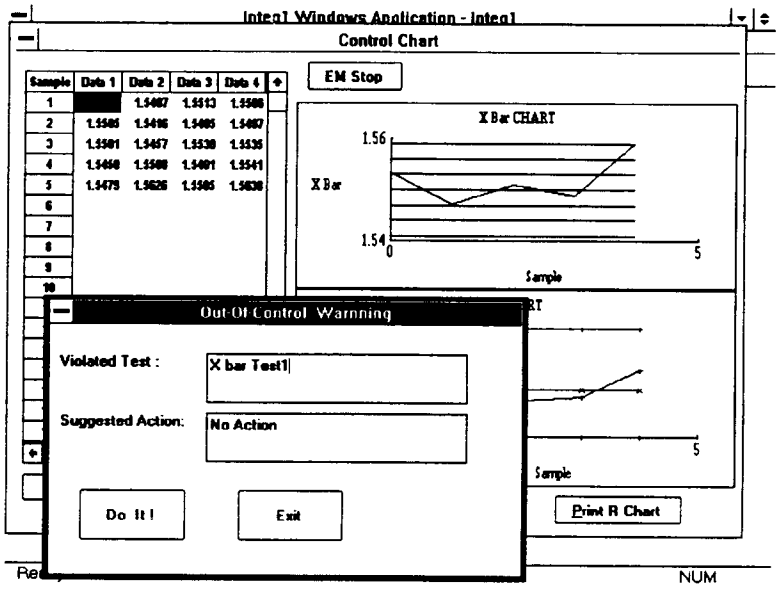


Figure 5-11 An Out-Of-Control Warning Dialog for $\overline{X}$ Chart Test 1 (Extreme Points)

5-4-4-2 Out of Control Signal Handling Implementation

When the process's out-of-control warning dialog pops up, the operation of the FMC is paused until the user clicks either "Do It" or "Exit" to resume the operation. If the user clicks the "Do It" button,

the suggested action for the out-of-control signal will be implemented. If the user clicks the "Exit " button, it exits the dialog without doing anything.

When a SPC test object has an embedded a response action to handle an out-of-control situation, the user can choose to execute this action if this out-of-control situation happens. For example, when the user clicks the "Do It" button in the Figure 5-12, the control computer shows a prompted message (as shown in Figure 5-13). The suggested action, a tool change, will be implemented in the next manufacturing cycle. After the worn out tool has been replaced, the process returns to normal as shown in Figure 5-14.

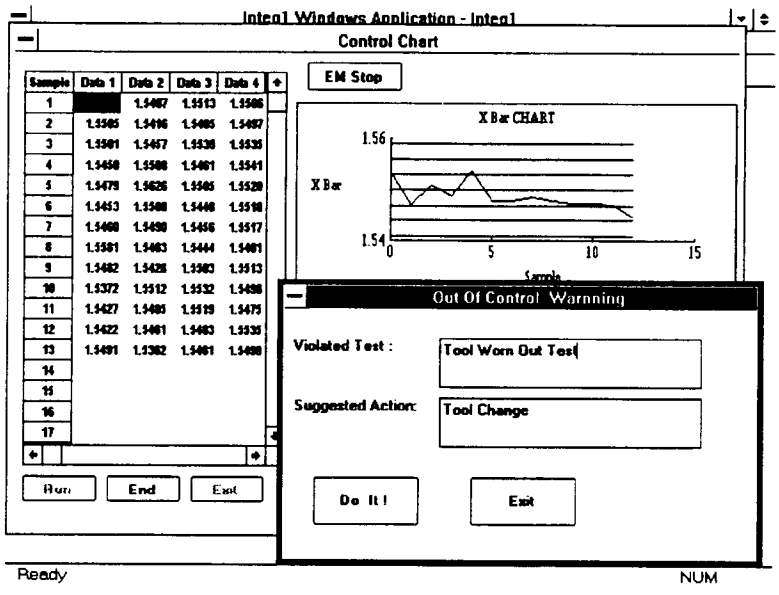


Figure 5-12 An Out-Of-Control Warning Dialog for Tool Wear Out Test

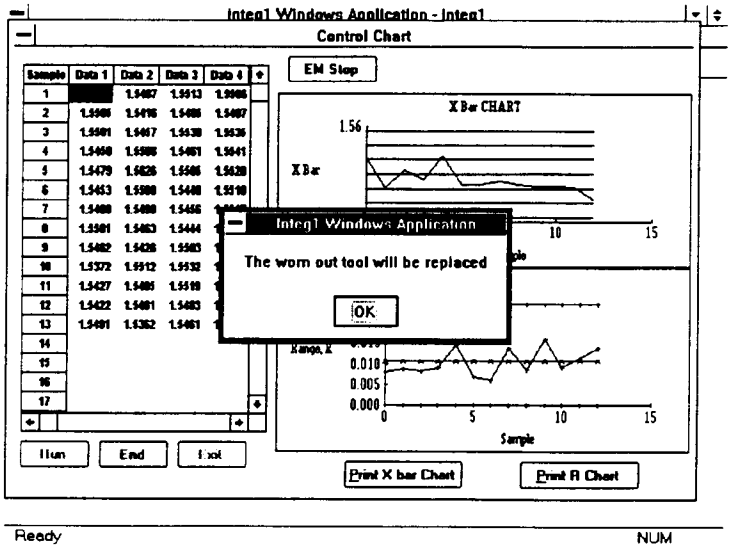


Figure 5-13 A Prompt Message in Response to "Do It" Action

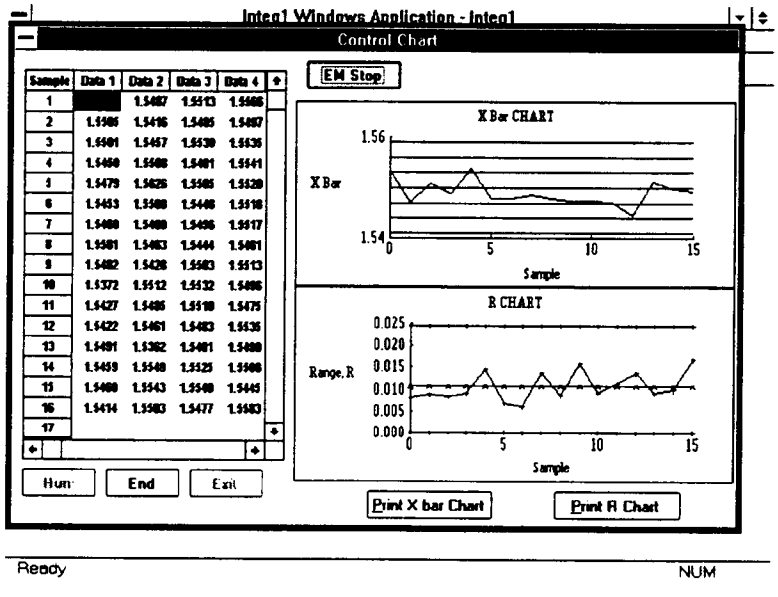


Figure 5-14 Results after the Worn Out Tool Is Replaced

5-4-5 System Validation

Two major tests were used in the final stage of the system development to test for the validation for integration and real-time SPC capability.

5-4-5-1 Integration Testing

The integration test focuses on testing the FMC's integrated operation, testing and monitoring of all the communicating input/output status. The "Input/Output Status" interface is used to test this integration operation. The final test results show that the integrated operation is functional as expected. Also, when the control computer sends a tool change command, the machining center uses an alternative tool for the manufacturing process.

5-4-5-2 Real-Time SPC Testing

For testing the real-time SPC function, a simulation application was developed to generate the sampling information and to send these data to the control computer via RS232 serial communication (Figure 5-15). These sample data formats are exactly the same as the real probing data format from the machining center.

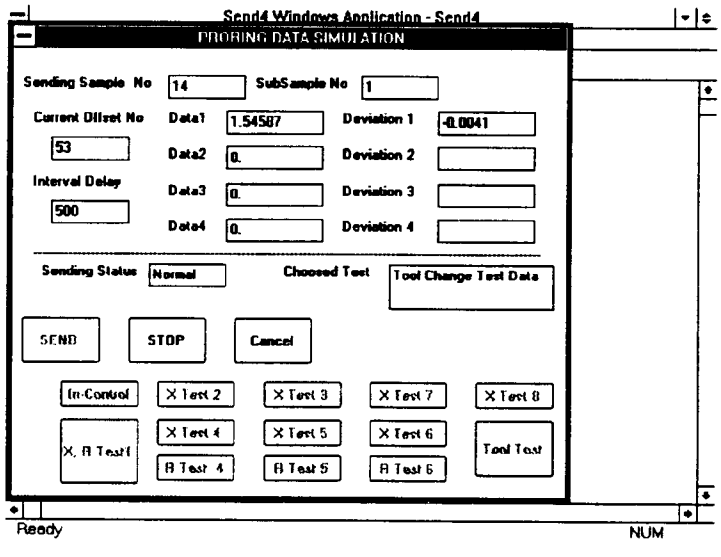


Figure 5-15 A Simulation Application for Real-Time SPC Test

There are thirteen different test data sets designed to test the $\overline{X}$ chart's eight tests, the R chart's four tests and one tool test. These test data sets are designed based on the testing part's diameter (shown in Figure 5-8) for generating the desired out-of-control signals. These test data sets are given in Appendix E.

During the real-time SPC tests, the control computer can receive all the sample data from a simulated computer, can display all received data in a two dimension table and plot $\overline{X}$ and R chart in real-time. Besides this, every out-of-control signal can be detected and the system will respond to an out-of-control warning dialog with the violated test message. The "Tool Test" SPC test shows that a special test can be designed for discovering a particular cause (tool's wear) for the process' out-of-control signal and the user can implement an action to recover the out-of-control process in real-time.

CHAPTER 6 CONCLUSION AND RECOMMENDATIONS

6-1 Conclusions

This thesis describes a development effort to design and implement strategies for computer-integrated flexible manufacturing with real-time SPC capabilities. The integrated model offers a top-down high level system design, modeling and testing procedure for integrated manufacturing. A scenario integrated table was used to model a manufacturing system while considering the interactive relationship among system components using Send(), Wait() and other high level functions. This table was transformed into Petri net models for more specific system testing and evaluation. After the final Petri net models were developed, high level operational programs were designed for operation and control of each DCS.

One of the most critical components in system development is its the communication system. The I/O Bus communication system offers the flexibility desired in integrating individual DCSs into a FMC. The Petri net models were used to ensure that the mutually exclusive requirement for the sending bus shared among DCSs is satisfied. Besides the I/O Bus, a serial input using an one bit input port is developed for the control computer to send messages to the HAAS machining center, while a RS232 probing and command message protocol is developed for the control computer to receive the messages from the HAAS machining center. These designs were required due to the HAAS machining center's communication limitation.

Sampling data from the machining center is used in process control. After the subgroup's $\overline{X}$ and R are calculated, they are plotted on the $\overline{X}$ and R control charts, respectively. An efficient algorithm is designed for real-time SPC process checking using the finite state mechanism. An object-oriented model is used to implement real-time SPC checking and out-of-control handling. It also has the capability of executing an embedded response action for an out-of-control signal in real-time.

6-2 Contribution

There are three main contributions of this research:

1. Integration Framework: An integration framework is developed and implemented. Such a framework is an effective way to design and model the real-time manufacturing system using several DCSs.
2. Real-Time SPC: The real-time SPC function can reduce error detection and analysis time because the underlying algorithm can perform SPC checks and detect out-of-control signals in a short period of time. In addition, the SPC application has the capability of implementing recovery actions when out-of-control conditions are detected. This SPC application can be used for other on-line measurement instruments via RS 232 communication.
3. Communication Interface: An I/O Bus is designed to not only integrate several DCSs, but also to directly control actuators. The design is flexible, easy to set up and economical. The object-oriented design for the I/O Bus makes future development or maintenance easy.

6-3 Future Enhancement

Several possible research extensions to this research are envisioned. These include:

- Development of a Design of Experiment Capability SPC Real-Time System: Design and implement sampling cost analysis, process capability assessment and on-line Design Of Experiment (DOE) for an FMC.
- Development of a Knowledge Based System: Extend this current system to develop a Knowledge Base System (KBS) for real-time SPC diagnosis. This KBS will respond to suggestions from the user. In addition, this system will be able to recognize the data pattern based on past experience.
- Development of a Multi-Tasking Real-Time SPC System: Implement a real-time multi-task SPC which includes several different sets of control charts, such as X, R_m , and Cusum charts and cause-and-effect diagrams.

- Development of an Object-Oriented Model for FMC: Identify characteristics of FMC's that can be independently operated and controlled, and develop functions for accomplishing these objectives; this will result in an object-oriented environment where modules can be used to develop a FMS.

REFERENCES

1. Andersen, John P., *Distributed Control System Testing*, ISA Transactions, Vol 30, No2, pp 41-5, 1991.

2. Black, J. T.; Schroer, B. J., *Decouplers In Integrated Cellular Manufacturing Systems*, Journal of Engineering for , Vol 110, pp 77-85, February, 1988.

3. Capkovic, F., *Petri Nets-Based Computer Aided Synthesis of Control Systems for Discrete Events Dynamic Systems*(Computer Aided Design in Control Systems- Edited by H.A. Barker), IFAC Symposia Series, No 1, pp 313-324, 1992.

4. Damsker, Dorel J., *Towards Advanced Concurrency, Distribution, Integration, And Openness Of A Power Plant Distributed Control System (DCS)*, IEEE Transactions on Energy Conversion, Vol 6, pp 297-302, June, 1991.

5. DeVor, Richard E. ; Chang, Tsong-how; Sutherland, John W., *Statistical Quality Design and Control.* Macmillan publishing ,1991.

6. Dicesare, F.; Harhalakis, G.; Proth, J.M.; Silva, M.; Vernadat, F.B., *Practice Of Petri Nets In Manufacturing*, Chapman & Hall, 1993

7. Farley, Richard, *Software Engineering Concepts*, McGraw-Hill, 1985.

8. Ferrarini, L.; Narduzzi, M.; Tassan-Solet, M., *A New Approach To Modular Liveness Analysis Conceived For Large Logic Controllers' Design*, IEEE Transactions on Robotics and Automation, Vol 10, pp 169-84, April, 1994.

9. Groover, Mikell P., *Automation, Production Systems, and Compter Integrated Manufacturing*, Prentice-Hall, pp482-483, 1987.

10. Irani, S. A.; Cohen, P. H.; Cavalier, T. M., *Design of Cellular Manufacturing* , Journal of Engineering for Industry, Vol 114, pp 352-61, August, 1992.

11. Knapp, Gerald M.; Wang, Hsu-Pin (Ben), *Modeling Of Automated Storage/Retrieval Systems Using Petri Nets*, Journal Of Manufacturing Systems (ISSN:0278-6125), Vol 11, No1, pp 20-9, 1992.

12. Kyde, William C.:3rd; Layden, John *Real-Time Data Acquisition Using SPC.* Manufacturing Engineering, Vol 101, pp 64-7 October , 1988.

13. Lee, J. L., *Application of Statistical Process Control Methods to The Establishment of Process Alarms in a Distributed Process Control System*, ISA Transactions, Vol 30 No 1, pp 59-64 , 1991.

14. Mamzic, C. L.; Tucker, T. W., *Incorporating Statistical Process Control Within A Distributed Control System,* ISA Transactions,Vol 30, No 1, p 11-24 , 1992.

15. Marsh, C. E.; Tucker, T. W., *Application of SPC Techniques to Batch Units, ISA Transation.* Vol. 28, No. 3, pp41-46, 1989.

16. Martin, J. M.: *Cells Drive Manufacturing Stategy,* Manufacturing Mengineering 102(1): pp 49-54, 1989.

17. Mitchell, F. H.; Jr. , *CIM SYSTEM- An Introdution to Computer-Integrated Manufacturing*, Prentice-Hall, 1991.

18. Nagakawa, Kazutaka; Matsudaira, Takayuki; Shiobara, Yasuhisa, *Distributed Control System Utilizing A MAP-Based LAN,* ISA Transactions, Vol 28, No 4, pp 47-55, 1989.

19. Nelson, L. S., *The Shewart Control Chart: Tests for Special Causes, Journal of Quality Technology*, Vol. 16, No 4 , pp. 237-239, 1984.

20. Noaker, Paula M., *Sizing Up SPC Software*, Manufacturing Engineering , pp. 32-50, April, 1995.

21. Pan T., *An Introduction of Distributed Control System*, AutoTech Magazine, Vol 129, pp 81-84, Jan., 1995.

22. Peterson, James L., *Petri Net Theory And The Modeling Of System*, Prentice-Hall, 1981

23. Ra, Shauo-chen, *Computer Integrating Manufacturing (CIM) CAD/CAM 's application*. Shong-Kon computer publishing, pp330, 1992.

24. Rembold, Ulrich; Armbruster, Karl; Ulzmann, Wolfgang, *Interface Technology for Computer-Controlled Manufacturing Process*, Marcel Dekker, 1983.

25. Rembold, Ulrich, Blume; Christian, Ruediger Dillmann, *Computer- Integrated Manufacturing Technology and System*, Marcel Dekker, 1985.

26. Rod , Michael G.; Deravi. Farzin, *Communication Systems for Industrial Automation*, Prentice Hall, 1989.

27. Scholz-reiter, B., *CIM Interfaces, Concepts, Standards And Problems Of Interfaces In Computer-Integrated Manufacturing,* Chapman & Hall, 1992.

28. Talavage, Joseph; Hannan, Roger G., *Flexible Manufacturing Systems in Pratrice* , Dekker, pp70, 1988.

29. Tanebaum, Andrew S., *Computer Networks*, Prentice-Hall International Editions, 1989.

30. Teng, Sheng-Hsien; Black, J. T., *Autonomous Cell Control System Design With Petri Nets*, Design, Analysis, and Control of Manufacturing Cells PED, Vol. 53, 1991.

31. Tsai, Jeffrey J.P; Yang, S. J., Chang, Y., *Timing Contraint Petri Nets And Their Application To Schedulability Analysis Of Real-Time System Specifications,* IEEE Transactions on Software engineering, Vol. 21, No 1, Jan. 1995.

32. Tsang, P. W. M.; Wang, R. W. C., *Development of a Distributive Lighting Control System Using Local Operating Network*, IEEE Transactions on Consumer Electronics, Vol 40, pp 879-89 November, 1994.

33. Wang, I-Peing, *Factory Digital Networking in Industrial Automation*, AutoTech Magazine, Vol 131, pp 110-119, March, 1995.

34. Yeager, R. L.; Davis, T. R. , *Reduce Process Variation With Real-Time SPC* Hydrocarbon Processing (International edition), Vol, 71, pp 89-92, March , 1992.

35. Yeager, Robert C., *Real-time SPC: Caveats And Capabilities,* PIMA Magazine, Vol 73 , pp 12, July, 1991.

36. Yeager, Robert, *Real-time Analysis and SPC,* PIMA Magazine, Vol 73, pp 12, May , 1991.

37. Yei, S., *The Newest National and Intelegent Standard for Field Bus* , AutoTech Magazine, Vol 124, pp 216-226, August, 1994.

38. Yei, S., *Process Control Machines and New Technologies*, AutoTech Magazine, Vol 132, pp 216-226 April, 1995.

39. Yim, Dong-Soon; Barta., Thomas A., *A Petri Net-Based Simulation Tool For The Design And Analysis Of Flexible Manufacturing Systems*, Journal of Manufacturing Systems , Vol 13, No4, pp 251-61, 1994.

APPENDICES

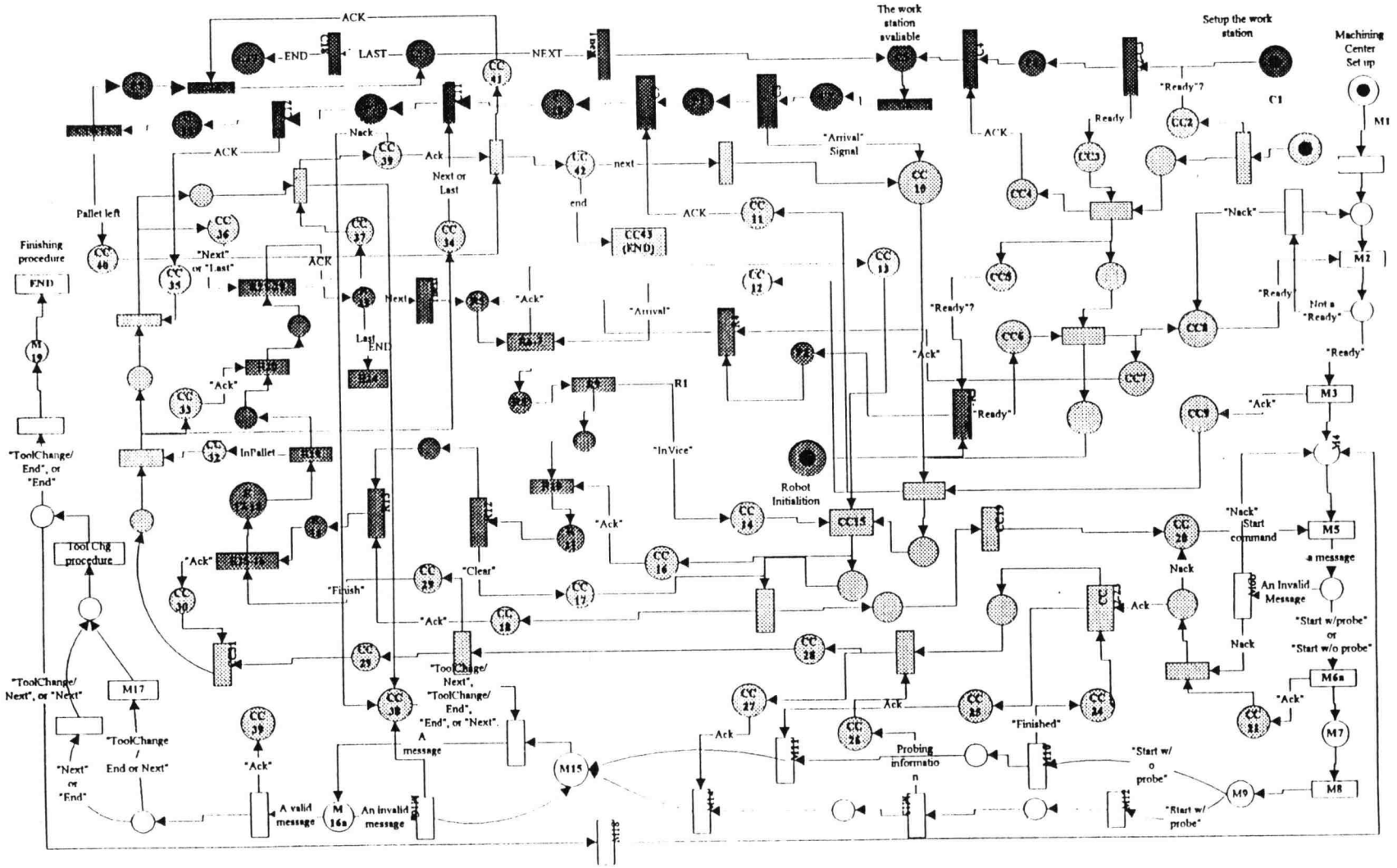


Figure A A Complete Integrated Petri Net Model for the FMC

APPENDIX B DCS OPERATIONAL PROGRAMS' FLOWCHARTS

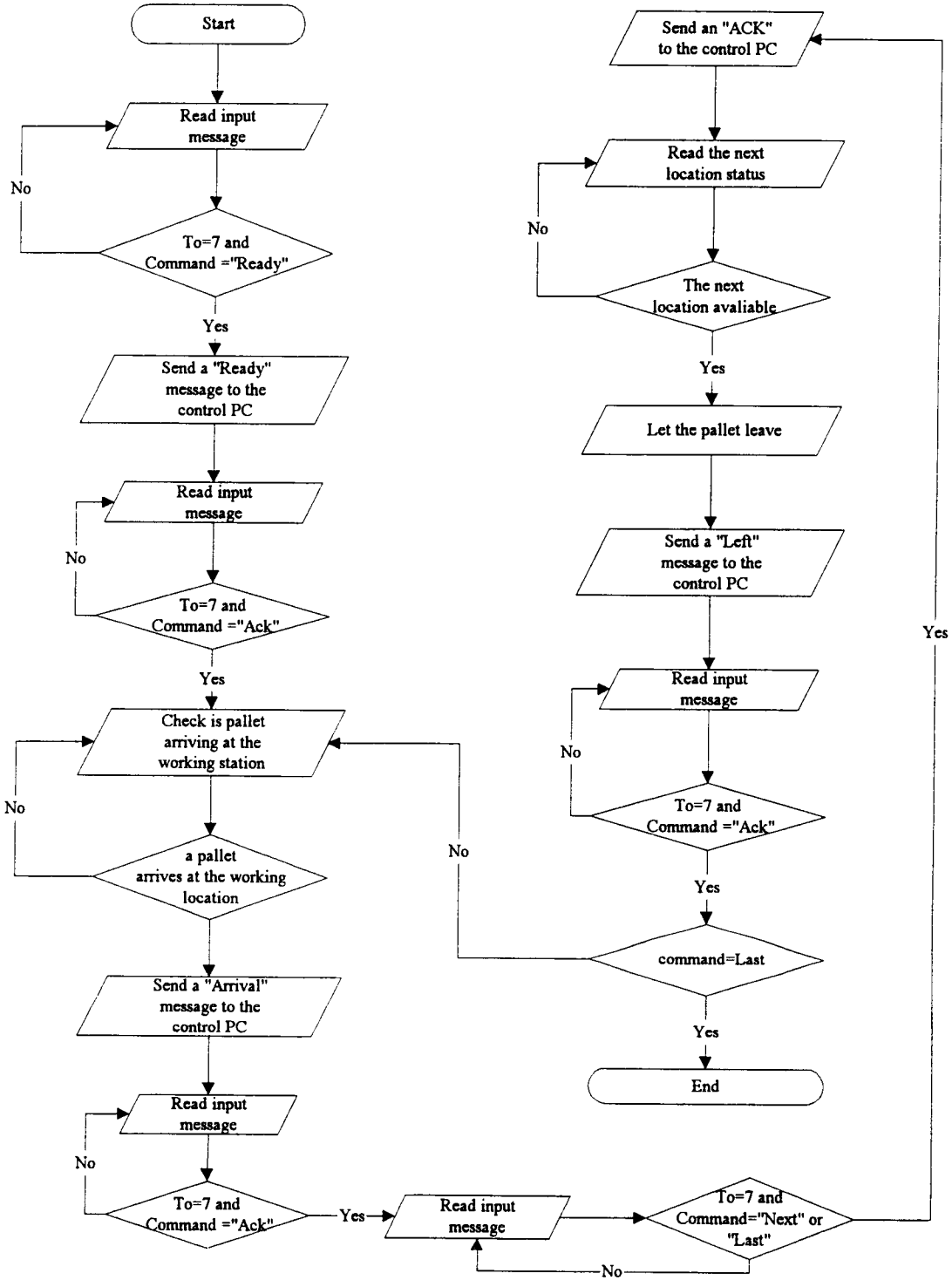


Figure B-1a Conveyor High Level Operational Program Flowchart (Level 1)

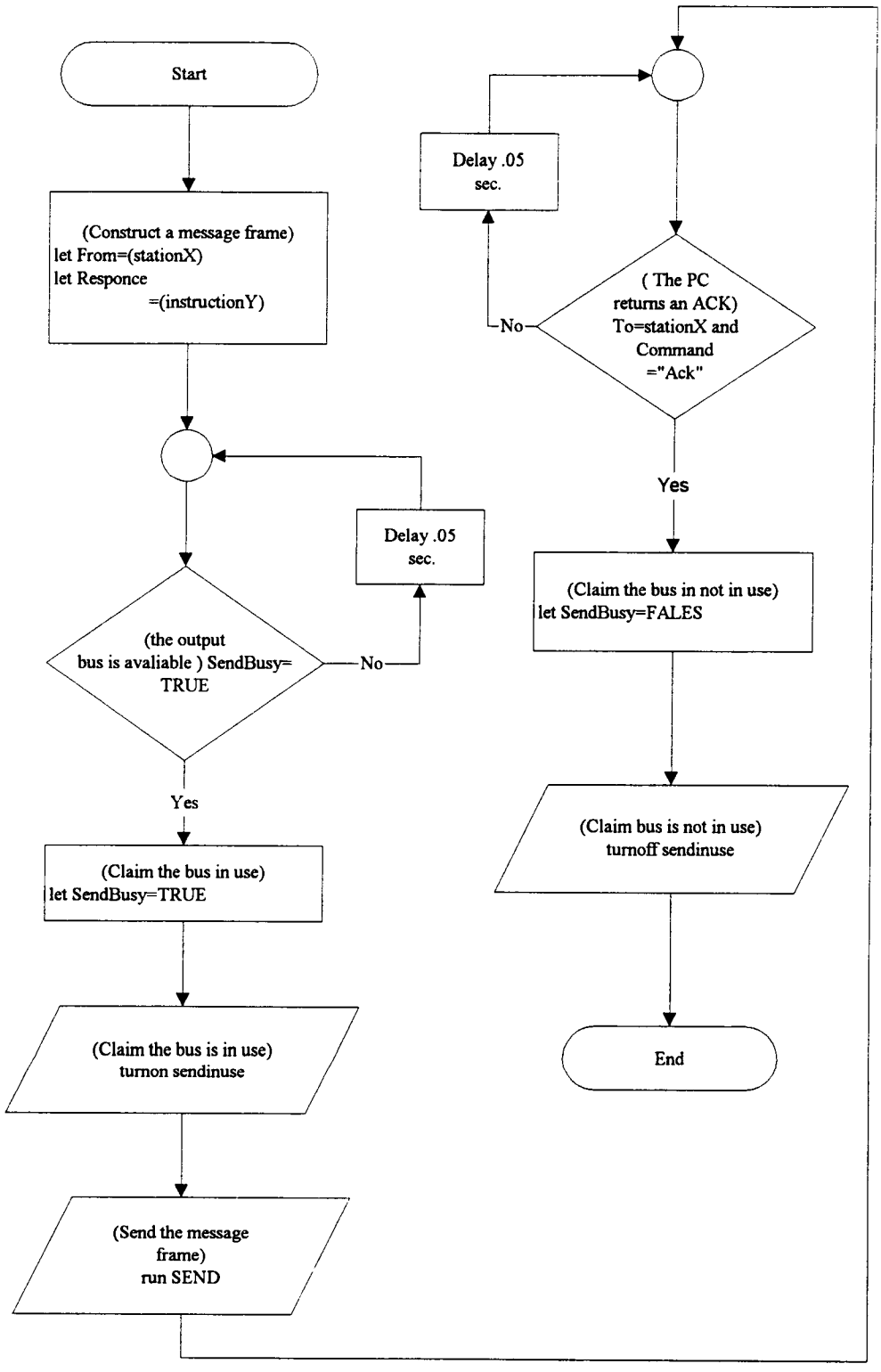
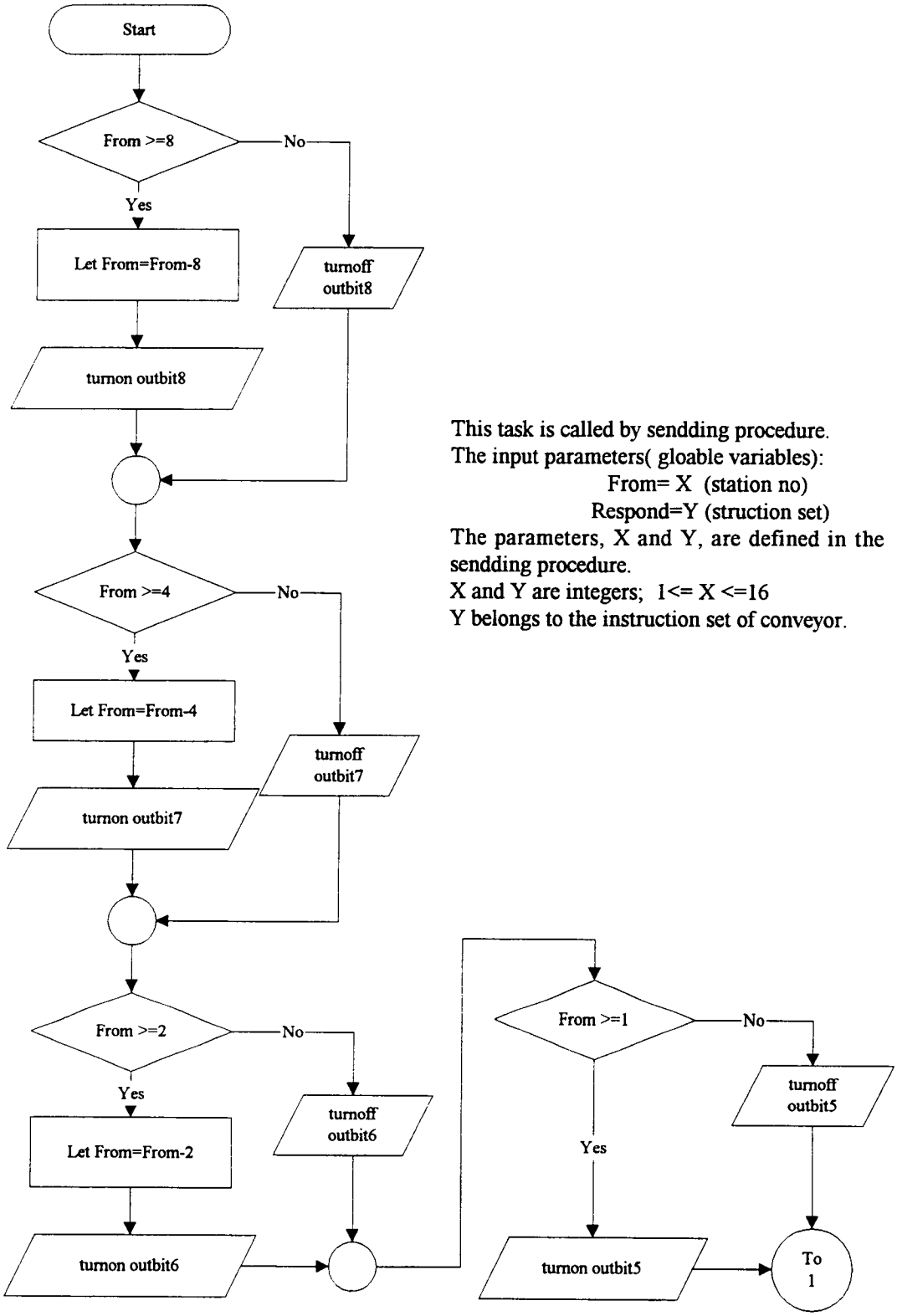


Figure B-1b Conveyor Sending Message Task Flowchart (level 2)



This task is called by sendding procedure.
The input parameters(gloable variables):
From= X (station no)
Respond=Y (struction set)
The parameters, X and Y, are defined in the sendding procedure.
X and Y are integers; 1<= X <=16
Y belongs to the instruction set of conveyor.

Figure B-1c Conveyor Send Message Task Flowchart (level 3)

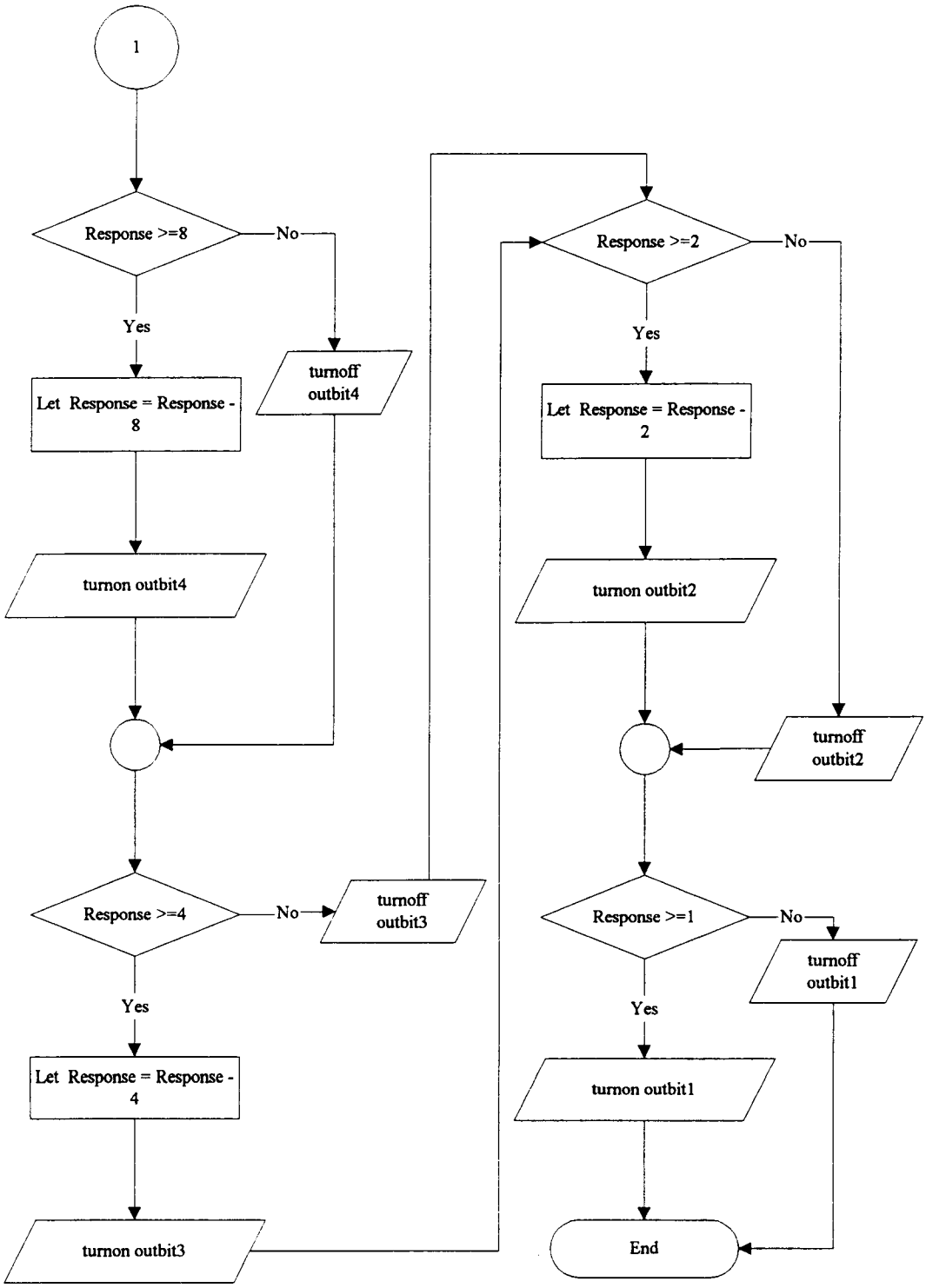


Figure B-1c Conveyor Send Message Task Flowchart (level 3) (continue)

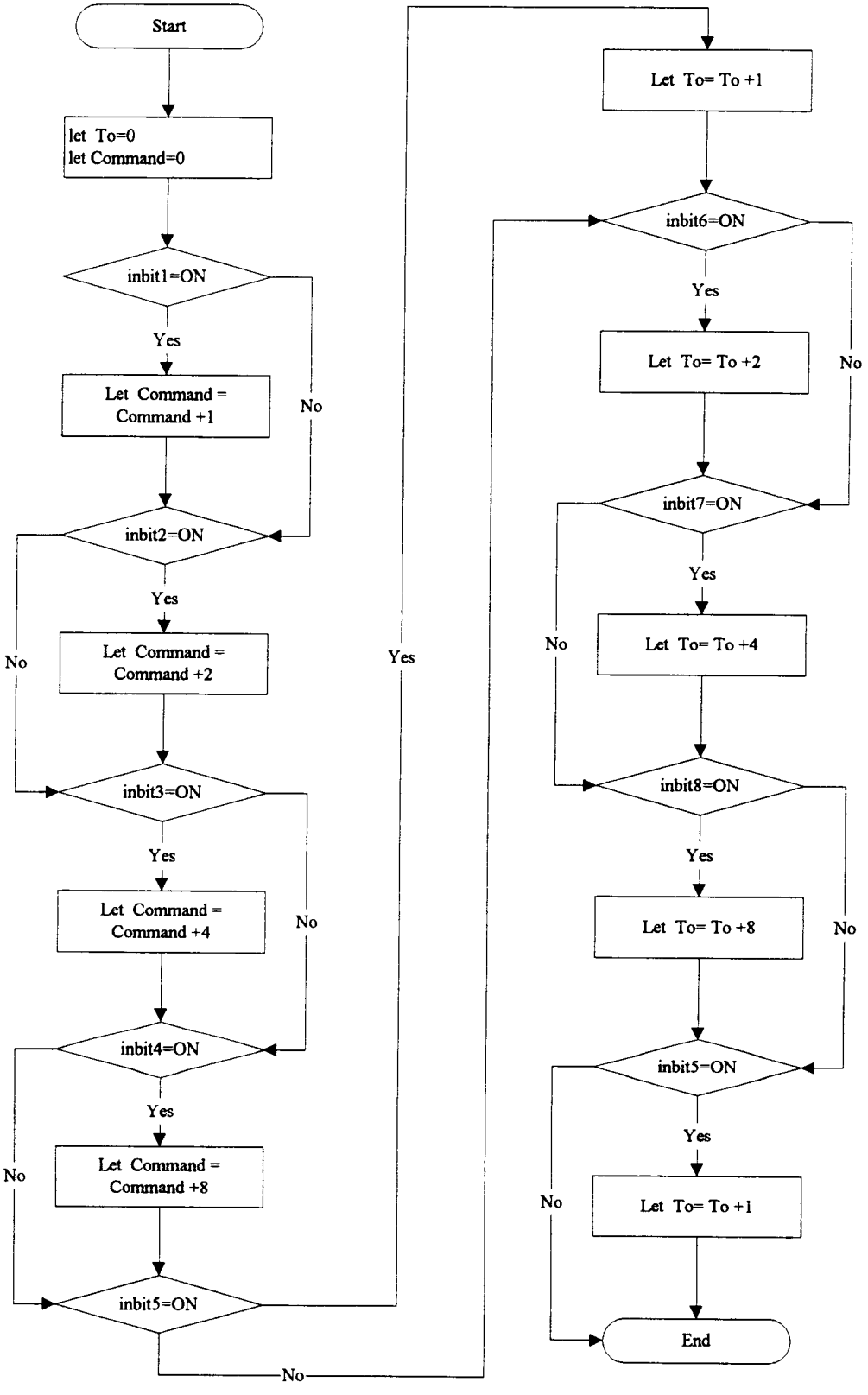


Figure B-1d Conveyor Receive Message Task Flowchart (level 2)

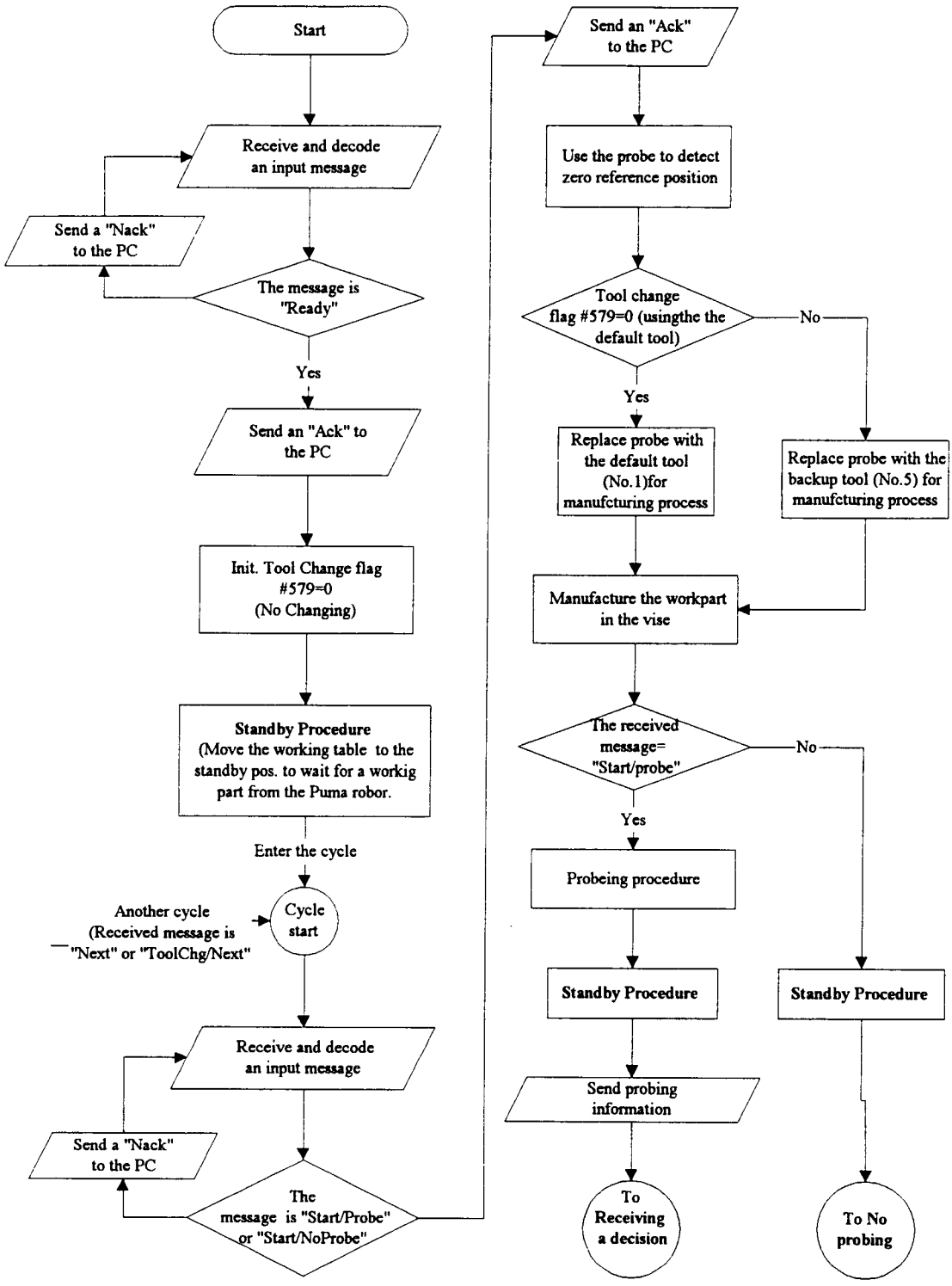


Figure B-2a The Machine Center CNC Program Flowchart

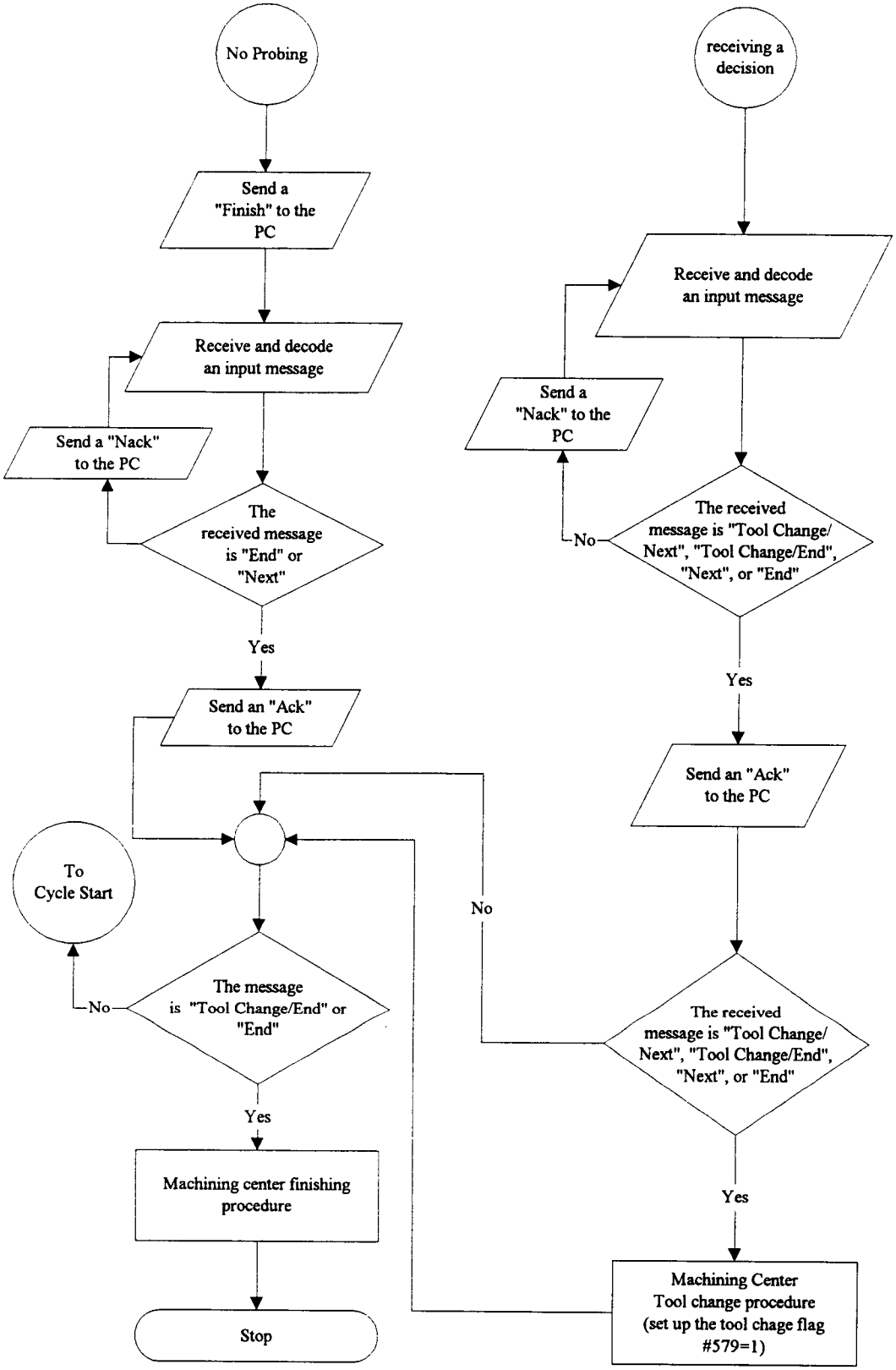


Figure B-2a The Machine Center CNC Program Flowchart (Continue)

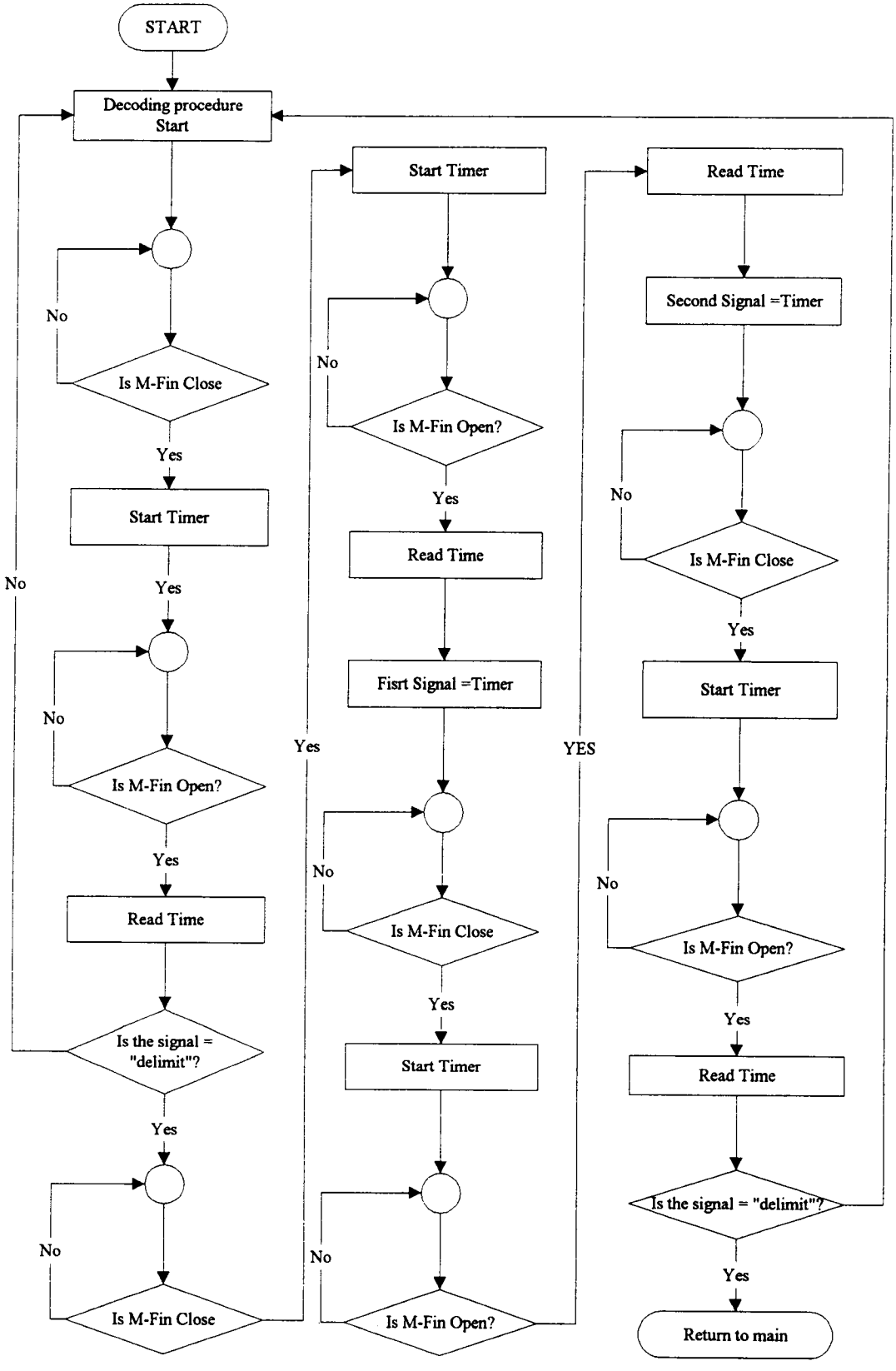


Figure B-2b Decoding Procedure Of The Machining Center

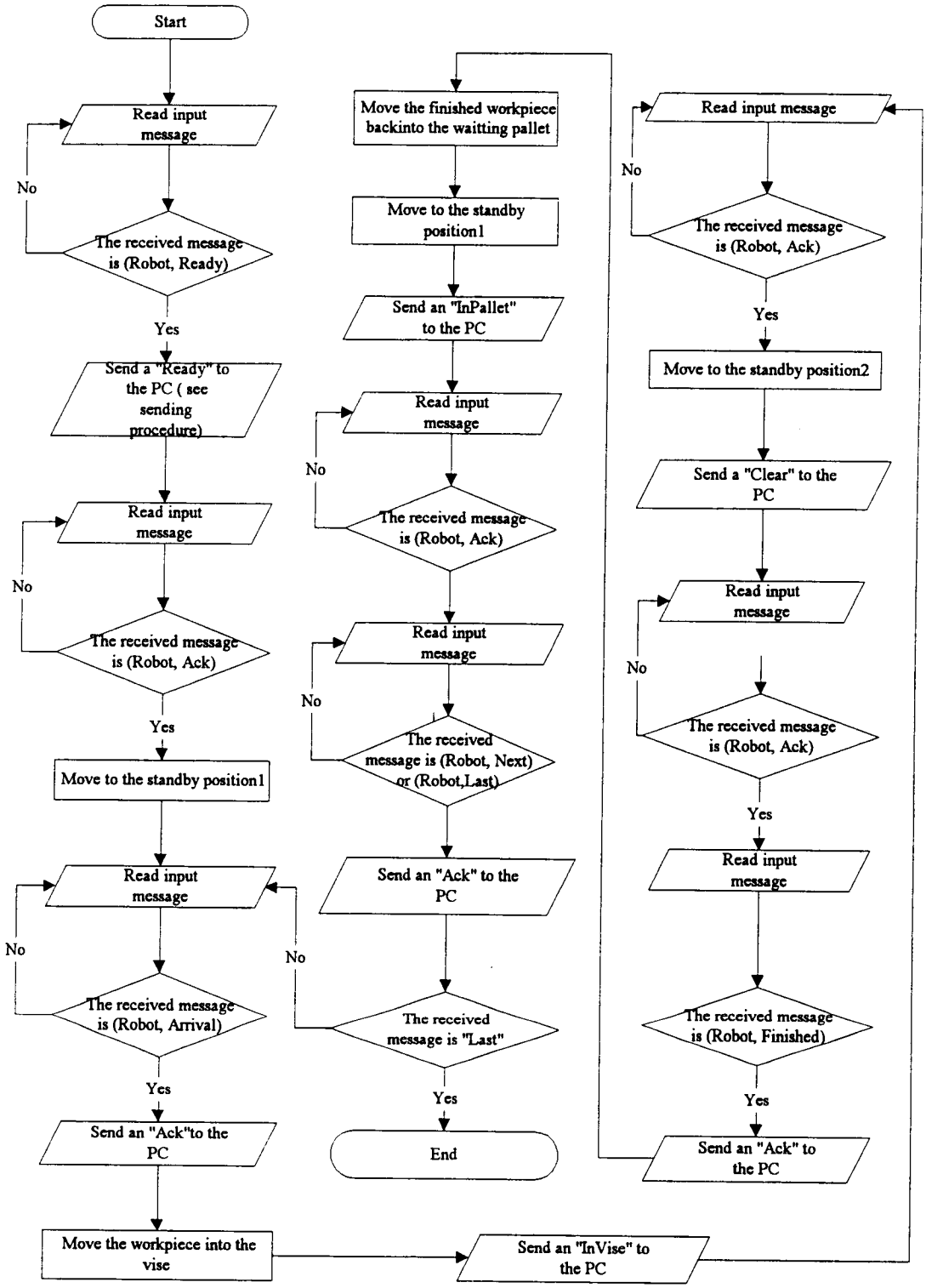


Figure B-3a Robot Program Flowchart (level-1)

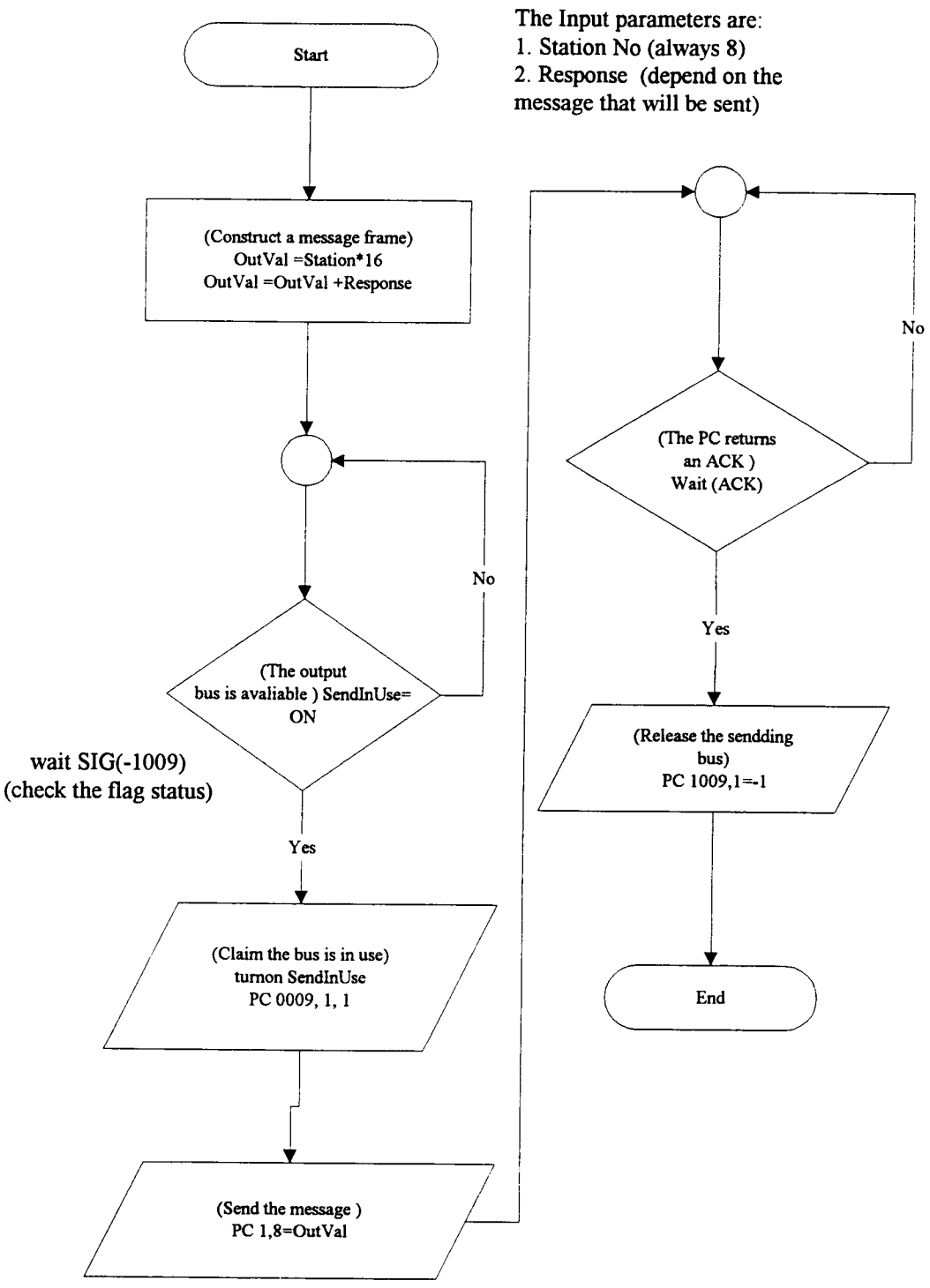


Figure B-3b The Robot Sending Message Procedure Flowchart (level 2)

APPENDIX C THE HIGH LEVEL CONTROL COMPUTER OPERATING PROGRAM

```
short CIOStatus::Procedure()
 {
 ////////////////////////////////////
 // Initialization          //
 ////////////////////////////////////
 short decision, chkflag;
 short decisionToMC;
 m_SampleNo=1;
 m_SubSampleNo=0;
 m_CurrentTotal=0;
 m_AmountNo=m_Amount;
 m_FreqNo=m_Freq;
 Send(0,0);
 OpenVise(); // Open the vise
 RS232();       // initalize the RS232 port
 ClearMsg(RecvMsg)
 ////////////////////////////////////
 // block #1
 // purpose   polling the Conveyor
 ////////////////////////////////////
 if (!m_StopFlag)
   if (CheckStep())
     {
     // execut the block commands
     Send(Conveyor,Ready);
     // Conveyor waits for the polling
     Wait(Conveyor,Ready);
     Send(Conveyor,Ack);
     // Conveyor is ready
     }
 DelayTime(1000);
 ////////////////////
 // block #2
 // purpose   polling the Robot
 ////////////////////////////////
 if (!m_StopFlag)
   if (CheckStep())
     {
     // execut the block commands
     Send(Robot,Ready);
     // ShowRobot("Wait for the polling");
     Wait(Robot,Ready);
     Send(Robot,Ack);
     // ShowRobot("Ready");
     }
 DelayTime(1000);
```

```
////////////////////////////////////////////////////
// block #3
// purpose polling the Machining Center
////////////////////////////////////////////////////
if (!m_StopFlag)
  if (CheckStep())
    {
    // execut the block commands
    // ShowMC("Wait for the polling");
    // ShowToMC("Sending a ready signal");
    DelayTime(200);
    MC.Send(MC_Ready);
    while ((!WaitRS232(ACK))&&(!m_StopFlag))
      {
      // The received command is Nack;
      //ShowToMC("Resending the ready signal");
      DelayTime(1500);
      MC.Send(MC_Ready);
      }

    // MC is ready");
    ShowToMC("    ");
    }
 DelayTime(1000);
///************************************************
// **The follwing is the manufactunging loop        **
// **                                               **
///************************************************
while ((!m_StopFlag)&&(!m_EndFlag))
  {
      ////////////////////////////////////////
      // Block #4
      // Purpose: Wait for a pallet's arrival
      //      It is the begining of the cycle
      ////////////////////////////////////////
      if (!m_StopFlag)
          if (CheckStep())
              {
              m_CurrentTotal++;
              // execut the block commands
              //ShowRobot("In Standby Position ");
              //ShowConveyor("Wait for a pallet's arrival");
              //ShowMC("At the start of the cycle");
              //ShowToMC("  ");
              ////////////////////////////////
              Wait(Conveyor,Arrival);
              //ShowConveyor("Pallet arrived at the working location");
              Send(Conveyor,Ack);
              DelayTime(2000);
              }


      DelayTime(1000);
```

```
////////////////////////////////////////////
// Block #5
// Purpose: Send the pallet arrival signal to the Robot
////////////////////////////////////////////
if (!m_StopFlag)
      if (CheckStep())
        {
        // execut the block commands
        //ShowRobot("Wait for the pallet's arrival");
        Send(Robot,Arrival);
        Wait(Robot,Ack);
        //ShowRobot("Ready to move the part to MC");
        //ShowConveyor("The pallet waitting part back");
        Wait(Robot, InVise);
        //ShowRobot("The part is in the vise");
        }


//DelayTime(1000);
////////////////////////////////////////////
// Block #6
// Purpose: Close the Vise
////////////////////////////////////////////
if (!m_StopFlag)
      if (CheckStep())
        {
        // execut the block commands
        CloseVise();
                DelayTime(9000);
                //m_ViseStatus="Closed" ;

        }


////////////////////////////////////////////////////
/// Block #7
/// Purpose: Make the robot move to standby position
////////////////////////////////////////////////////
if (!m_StopFlag)
      if (CheckStep())
        {
        // execut the block commands
                Send(Robot,Ack);
                DelayTime(2000);
        //ShowConveyor("The pallet waitting part back");
        //ShowRobot("Moving to the Clear position");
        Wait(Robot,Clear);
        //ShowRobot("Wait for the part being finished");
        Send(Robot,Ack2);
        //ShowMC("Waiting for a start signal");
        }
```

```
////////////////////////////////////////////////
// Block #8
// Purpose: Make a decision for manufacturing the part with
//              or without probing
////////////////////////////////////////////////
if  ((m_CurrentTotal % m_Freq)==0)
        decisionToMC=Start_Probe;
else
        decisionToMC=Start_NoProbe;
////////////////////////////////////////////////
// Block #9
// Purpose: Send a start signal to MC and wait for
//       a finished signal from the MC
////////////////////////////////////////////////
if (!m_StopFlag)
        if (CheckStep())
        {
        // execut the block commands
        //ShowMC("Waiting for a Start/Probe signal");
        MC.Send(decisionToMC);

        while ((!WaitRS232(ACK))&&(!m_StopFlag))
                {
                // The received command is Nack;
                //  ReSending a "Start/NoProbe" or "Start/Probe" signal
                MC.Send(decisionToMC);
                DelayTime(2000);
                }

        // ShowMC("Manufacturing the working part.");
        ////////////////////////////////////////////
         // Wait for finished signal or probing information

        ////////////////////////////////////////////
        if (decisionToMC==Start_NoProbe)
                {
                //ShowMC("Ready to send finished if part finished");
                DelayTime(1500);
                WaitRS232(FINISHED);
                // no error checking here !!!!
                 }
        else    // decision!=Start_NoProbe
         ////////////////////////////////////////////
                {
                        // ShowMC("Ready to send a probing information");
                        DelayTime(1500);
                        WaitProbe();
                }
        //  Part is finished
        }
DelayTime(1000);
```

```
////////////////////////////////////////////
// Block #10
// Purpose: Open the Vise
////////////////////////////////////////////
 if (!m_StopFlag)
        if (CheckStep())
          {
          // execut the block commands
          OpenVise();
                  DelayTime(9000);
                  //m_ViseStatus="Opened" ;
          }
 DelayTime(1000);

////////////////////////////////////////////////
/// Block #11
/// Purpose: Make the robot move the part back to
///       the waiting pallet
////////////////////////////////////////////////
 if (!m_StopFlag)
        if (CheckStep())
          {
          // execut the block commands
          Send(Robot,Finished);
          //ShowRobot("Moving the finished part back to the pallet");
          Wait(Robot,Ack);
          }

////////////////////////////////////////////////////
/// Bolck #12
/// Purpose:  Make a decision for M.C. and plot and make
///       control chart if necessary.
////////////////////////////////////////////////////
DelayTime(1000);
        if (m_CurrentTotal >=m_Amount)
                m_EndFlag=TRUE;
        if (m_EndFlag)
                decision=Last;
        else
                decision=Next;

        if (decisionToMC==Start_Probe)
                {

                m_SubSampleNo++;
                if (m_SubSampleNo >=m_Size)
                        {
                        CalculateXbarR();
                        WriteToGrid1();  // write the current subsample data into the
                                         // the dimension grid [m_SampleNo][m_SubSample]
                        WriteToGrid2();  // write X bar aand R into the grid
                        PlotToCharts();
                        SPCTests();
```

```
                    // tool changeflag set up and anf out-of-control tests  saving to the
                    //database
                    // are accomplished in SPCTests
                    ////////////////////////////////////////////
                    SaveData();   // save Sub Sample data
                    SaveSample(); // save X bar, R data
                    ////////////////////////////////////////////
                    m_SampleNo++;
                    ////////////////////////////////////////////
                    //===== It is for test version only. ========
                  if (m_SampleNo >45)
                      m_EndFlag=TRUE;

                    ////////////////////////////////////////////
                    m_SubSampleNo=0;
                    } // if (m_SubSampleNo >=m_Size)
                  else
                    {
                    WriteToGrid1();  // write the current subsample data into the
                                     // the dimension grid m_SampleNo][m_SubSample]
                    ////////////////////////////////////////////
                    SaveData(); //  SubSample data
                    ////////////////////////////////////////////

                    } // end of else  (m_SubSampleNo >=m_Size)

          }      // end of  if (decision==Start_Probe)
                 // ---------------------------------------------
////////////////////////////////////////////////////////////////////////////////////////////
 /////  The inplementation of the task for the violated tests ////////////////
////////////////////////////////////////////////////////////////////////////////////////////
     // make up some data
 if (m_ToolTest->m_ActionFlag)
        {
        m_ToolChgFlag=TRUE;
        m_ToolTest->m_ActionFlag=FALSE;
        MessageBox("The worn out tool will be replaced");
        }
////////////////////////////////////////////
 if ( m_ToolChgFlag==TRUE)
        if ( decision==Last)
                decisionToMC=ToolChg_End;
        else
                decisionToMC=ToolChg_Next;
        else
                if ( decision==Last)
                        decisionToMC=MC_End;
                else
                        decisionToMC=MC_Next;
```

```
//////////////////////////////////////////
/// Block #13
/// Purpose: Wait for the part back into the waiting pallet
//////////////////////////////////////////
      if (!m_StopFlag)
        if (CheckStep())
          {
          // execut the block commands
          Wait(Robot,InPallet);
          //ShowRobot("In standby position");
          Send(Robot,Ack);
          DelayTime(2000);
          }
            DelayTime(1000);

//////////////////////////////////////////
/// Block #14
/// Purpose: Send a "Next" or "Last" to the conveyor.
//////////////////////////////////////////
 if (!m_StopFlag)
       if (CheckStep())
         {
         // execut the block commands
         if (decision==Last)
           Send(Conveyor,Last);
         else
            Send(Conveyor, Next);
         Wait(Conveyor,Ack);
         //ShowConveyor("The pallet is leaving");
         }
DelayTime(1000);

//////////////////////////////////////////
/// Block #15
/// Purpose:  Send the decision to the robot
//////////////////////////////////////////
if (!m_StopFlag)
       if (CheckStep())
         {
         // execut the block commands
         if (decision==Last)
           Send(Robot,Last);
         else
           Send(Robot,Next);
         Wait(Robot,Ack);
         }
DelayTime(1000);
```

```
////////////////////////////////////////////////
// Block #16
// Purpose: Send the decision(Tool Change/Next, ToolChange
//          / End, End or Next) to the MC
////////////////////////////////////////////////
//decision=MC_Next;
 if (!m_StopFlag)
        if (CheckStep())
          {
          // execut the block commands
          ShowMC("Waiting for a decision signal");
          switch (decisionToMC)
            {
            case MC_Next:
               //ShowToMC("Sending a Next signal");
               MC.Send(MC_Next);
               break;
            case MC_End:
               //ShowToMC("Sending an End signal");
               MC.Send(MC_End);
               break;
            case ToolChg_Next:
               //ShowToMC("Sending a ToolChange/Next signal");
               MC.Send( ToolChg_Next);
               break;
            case ToolChg_End:
               //ShowToMC("Sending a ToolChange/End signal");
               MC.Send( ToolChg_End);
               break;
            } // end of switch

          chkflag=FALSE;
          while ((!m_StopFlag)&&(!chkflag))
                    {
                    chkflag=WaitRS232(ACK);
                    //The received command is Nack;

                    ShowRS232Msg(RecvMsg);
                    if (!chkflag)
                              {
                              //ShowToMC("Resending decision signal");
                              DelayTime(2000);
                              MC.Send(decisionToMC);
                              }
                    } // end of while
        // MC is ready for another cycle
        }
DelayTime(1000);
```

```
        ////////////////////////////////////////////
        // Block #17
        // Purpose: Wait for a pallet's leaving signal
        ////////////////////////////////////////////
        if (!m_StopFlag)
                if (CheckStep())
                  {
                  // execut the block commands
                  //ShowConveyor("The pallet is ready to leave");
                  Wait(Conveyor,Left);
                  //ShowConveyor("The Pallet left");
                  Send(Conveyor,Ack2);
                  DelayTime(1000);
                  }
} // end of while loop
////////////////////////////////////////////
//   The end of the procedue
////////////////////////////////////////////
m_StopFlag=TRUE;
TSMCloseComm( m_ComId );
return TRUE;
}
```

APPENDIX D COMMUNICATION INTERFACING DESIGN AND IMPLEMENTATION

D-1 Overview

The objective of the communication system design in this research is to use a communication system to link all the DCSs. The communication system's range includes layers 4 and 5 in Figure 2-3 and the fieldbus concept is applied in the design of this system. However, due to the limitation of the specific control software of the physical component of the system used here, this research can only apply advance concepts in the low speed SSR-24 I/O board. Furthermore, the HAAS machining center's worst feature is that it is not designed to support interfacing with other machines. A special design for communication between the control computer to the MC was developed to solve this problem. To test the communication functions, a test interface function was developed as discussed in Section D-3.

D-2 I/O Bus Design For Integrated Control Systems

The I/O bus is designed to communicate between the control computer and the conveyor as well as the control computer and the robot. This bus supports Send (station, command) and Wait(station, command) functions for the integration of the FMC. Every parameter (stations and commands) must be clearly defined and assigned an integer number (see Tables D-1 and D-2).

The I/O bus architecture design is shown in Figure D-2. This I/O bus can support up to 15 workstations and 15 different messages (see Tables D-1 and D-2) or 15 * 15 = 225 revolutions (the maximum capacity for transmission) using 8 input Solid State Relays (SSR) and 8 output SSRs. If the traditional interlock method to implement this communicating function id used, it needs 255 input SSRs and 255 output SSRs. Therefore, the I/O bus design's cost is only 3.56 % of the traditional cost ((8 input SSRs + 8 output SSRs) /(225 input SSRs + 225 output SSRs)). The I/O bus is both easy to install and to maintain. For example, the set up procedure for a new workstation (assuming it already has the

same specifications- 8 inputs and 8 outputs SSRs) only needs to connect its I/O bus to the input and output bus of the I/O bus of the original system besides communication protocols' set up.

Table D-1 Interpretation Table Of Signal For Station Definition in Send() and Wait()

STATION		STATION		STATION	
No	Name	No	Name	No	Name
1		6		11	
2		7	Conveyor Working Location	12	
3		8	Puma Robot	13	
4		9		14	
5		10		15	

Table D-2 Interpretation Table Of Signal For Command's Definition in Send() and Wait()

Command No	Conveyor	Computer	Robot
1	Ready	Ready	Ready
2		Part In Vise	Part In Vise
3	Part Arrival	Part Arrival	Part Arrival
4		Clear	Clear
5	Ack	Ack	Ack
6		StandBy	StandBy
7	Next	Next	Next
8	Last	Last	Last
9	Left	Left	
10		Part In Pallet	Part In Pallet
11		Finished	Finished
12	Ack2	Ack2	Ack2
13			
14			
15		Nack	Nack

Because the I/O bus, a limited resource, needs to be shared by each DCS, mutually exclusive is necessary. Petri nets are used to ensure this requirement.

D-2-1 I/O Bus Petri Nets Model

A sending bus of the I/O bus is designed using a Petri net model. The design of the sending bus system in Figure D-1 and Table D-3 makes sure that the sending bus is used exclusively. The exclusivity

of the sending bus must be implemented; otherwise two or more different messages can be mixed together causing communication errors.

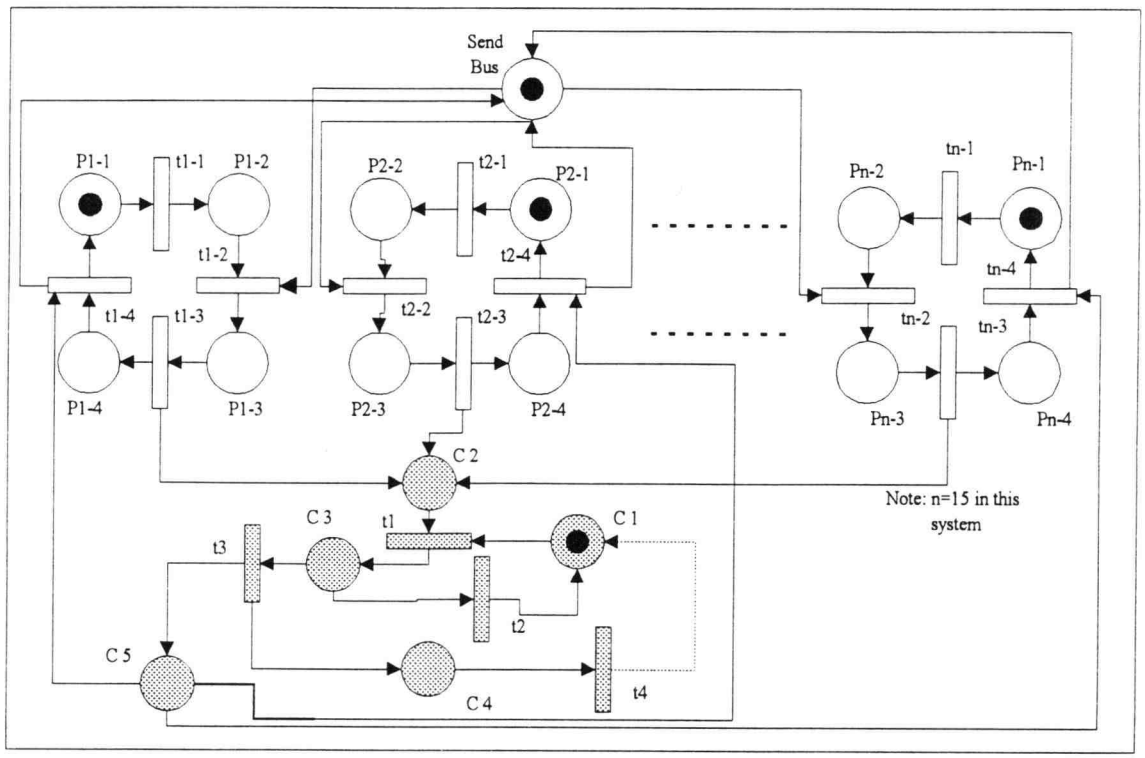


Figure D-1 A Petri Net Model Of The Sending Bus

The token in the place "Send Bus" (Figure D-1) represents the usage of a sending bus. The following description describes an example of the workstation 2 sending a message to the control computer.

1. Initially, a token is in place p2-1. If the workstation 2 needs to send a message to the control computer, it will enable the transition t2-1.
2. The transition t2-1 fires. This means that a sending message frame, a Application Protocol Data Unit (APDU), is constructed.
3. When a token flows into place-p2-2, the workstation checks the sending bus usage. If a token is in the Send Bus place, it means the sending bus is available. The transition t2-3 is enabled.

4. The firing of transition t2-3 represents the sending of message to the control computer. After t2-3 fires, the token in Send Bus place is removed. This means that no other workstation can use the sending bus until the sending bus is released by workstation 2.
5. After the control computer receives the sent message (a token to C2), it returns an Ack (a token from C5 to t2-4). In the Petri net model, the C5 with a token and p2-4 with a token will enable the transition t2-4.
6. In the Petri net, when t2-4 fires, a token will be sent back to place "Send Bus". This means that the sending procedure is completed and the workstation releases the sending bus's usage right.

Table D-3a Explanation Of The Sending Bus Petri Nets (for the ith DCS)

Conditions	Events
Pi-1 Need to send a message to the PC	Ti-1 Construct the sending message frame
Pi-2 Check the sending bus availability	Ti-2 Claim the sending bus in use
Pi-3 Enabling the sending bus	Ti-3 Send the message to the PC
Pi-4 Check if PC's ACK arrived	Ti-4 Release the Sending bus.

Table D-3b Explanation Of The Sending Bus Petri Nets (for the Control PC)

Places	Conditions	Transitions	Events
C1	Waiting for the incoming message	t1	Decode the received message
C2	Receive a message from a DCS	t2	Delay .10 second
C3	Check if the message is valid or invalid? Check twice.	t3	Construct an ACK sending frame and the PC continues its operation
C4	Other conditions in the PC operations	t4	Other events in the PC operation
C5	Sending the ACK message to the DCS		

D-2-2 I/O Bus Structure

The I/O bus is designed using eight input relays to construct an input bus and eight output relays to construct an output bus (see Figure D-2). In the I/O bus design, when the control computer is sending a message to the destination, every workstation is listening. A work station only "picks up" its message and neglects all others. But there is only one workstation that can send its message to the control computer at a time. The other workstations who want to send their message must wait for the current user to release the sending bus.

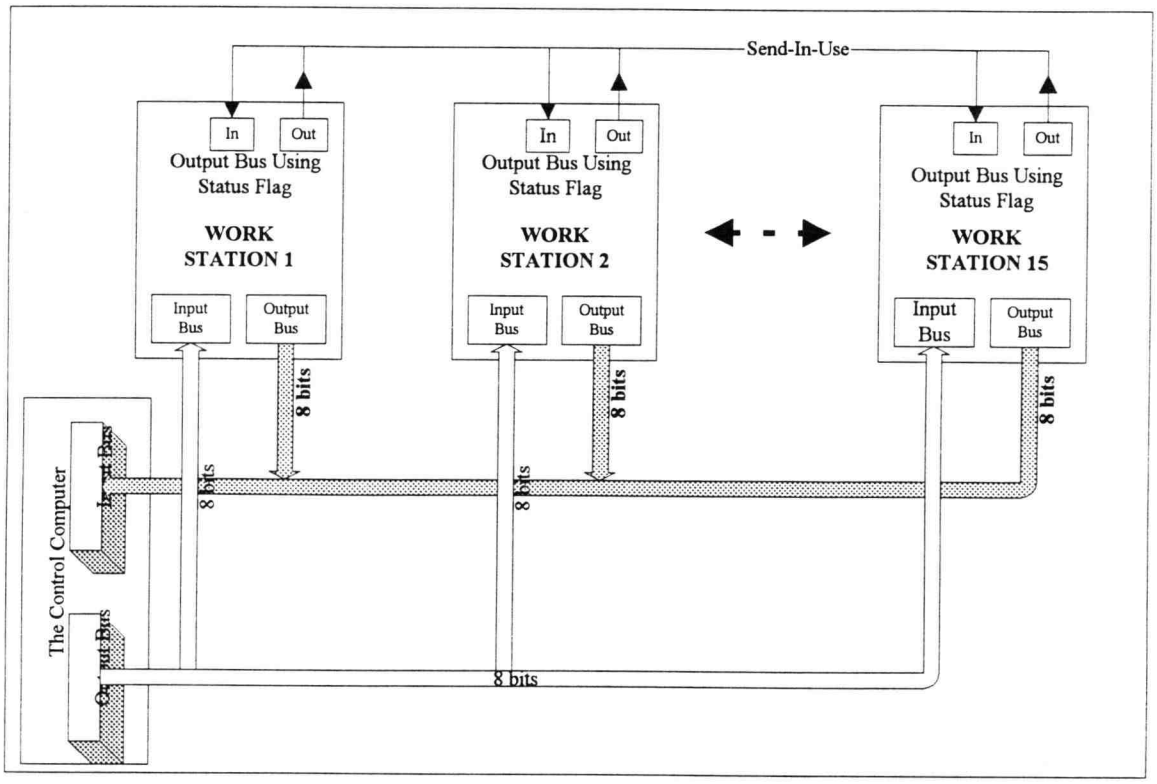


Figure D-2 I/O Bus Architecture

If a DCS needs to send a signal to the control computer, it first needs to check whether its Send-In-Use (1-bit input relay) is on or off. If it is on, it means the sending bus is in use; therefore, it just keeps waiting until the station in current use releases the bus and turns off the send-in-used. When the waiting station is going to use the sending bus, it turns on Send-In-Use to claim the sending bus is occupied. In this research, only two workstations employ the I/O bus; however, another station can be very easily added onto this I/O bus structure in future.

D-3 Serial Input by One Bit Input Port and RS 232 Probing Message and Command Handling

It was very difficult and time consuming to design a communication system for the HAAS MC, because it only supports one bit input and 3 output relays. The machining center's timer and the one bit input port are used to receive a serial input pulse (like the old fashion telegram) from the control computer and to decode the received signal to instructions. The MC uses RS232 serial communication

to send ASCII strings of probing information and commands instead of using the MC's three output relays.. A RS232 decoding protocol has been designed to receive the message from MC and decode it from ASCII code into a message frame.

D-3-1 Serial Input by One Bit Input Port Protocol Design

A signal from the PC to MC has been constructed with the following format.

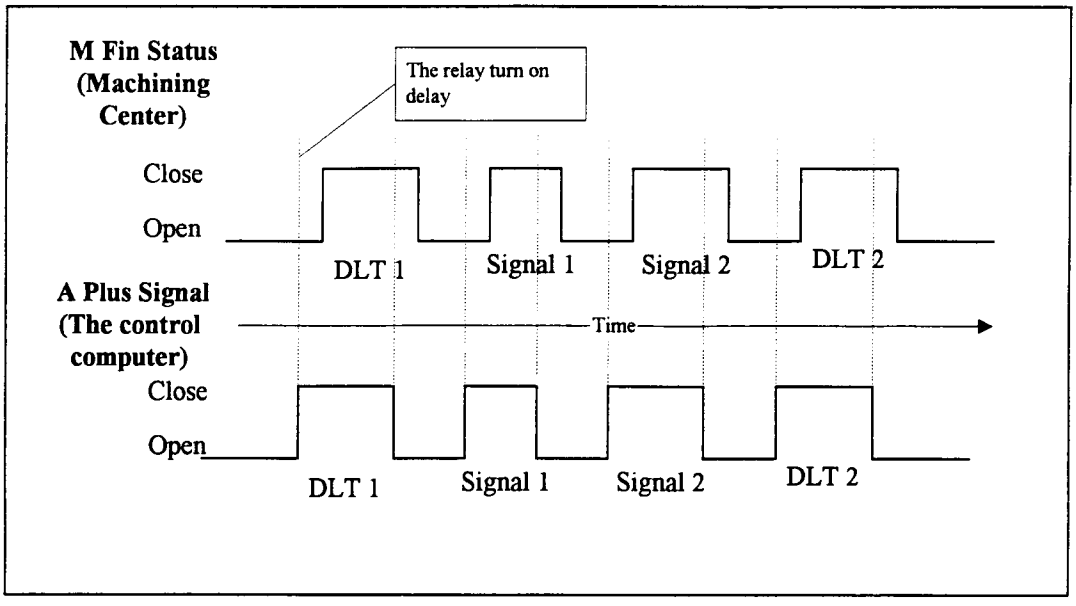


Figure D-3 A Serial Pulse Message Signal

There are three different types of signals: short, medium, and long. They are defined by the length of duration.

- Signal = Short when 0 < Signal's Duration <= a where a is a cut off value
- Signal = Medium when a < Signal's Duration <= 2a
- Signal = Long when 2a < Signal's Duration <= 3a

When the machining center receives a short signal, the $Signal_x$ is 0 where x is 1 or 2. When the machining center receives a medium signal, the $Signal_x$ is 1 where x is 1 or 2. When the machining center receives a long signal, the $Signal_x$ is 2 where x is 1 or 2. The machine center's protocol can decode a received message into a number from 0 to 8 using. Decoded Value = $Signal_1$ *3 + $Signal_2$

The mapping table for the decoded value to the instruction from the computer is given in Table D-4.

Table D-4 Interpretation Table Of Serial Input by One Bit Input Port Signal For Machining Center

Code Value	Code Meaning	First Signal		Second Signal		Comment		
		Type	Duration	Type	Duration		Desire	Cut-off
0	Ack	Short	0.35	Short	0.35	Short	0.35	0.7
1	Next	Short	0.35	Medium	1.05	Medium	1.05	1.40
2	Ready	Short	0.35	Long	1.75	Long	1.75	2.1
3	ToolChg/Next	Medium	1.05	Short	0.35	DLT		
4	Start w/probe	Medium	1.05	Medium	1.05	Nom.	1	
5	ToolChg/End	Medium	1.05	Long	1.75	Tol.	0.05	
6	Start w/o probe	Long	1.75	Short	0.35	Upper	1.05	
7	End	Long	1.75	Medium	1.05	Lower	0.95	
8		Long	1.75	Long	1.75	Interval	0.5	

D-3-2 RS 232 Probing Message and Command Handling

There are two kinds of messages from the MC to the control computer. One is probing information whose format is " offset Nominal Deviation". For example, "45 1.5500 -0.019" means that a sample's offset is 45, nominal is 1.5500, and deviation is -0.019. This probing data format is designed by Renshaw, the probe's dealer. The other message is command formats. For example, "COMMAND FINISHED" is one of the command strings.

A window based RS232 communication protocol is designed for this research. Windows communication Application Programming Interface (API) insulates some of the details of serial communication programs, such as DOS's RS232 programming. Furthermore, the RS232 protocol for this research is developed in a more user friendly and efficient manner so that it can be handled using some Window API function calls. This RS232 protocol can handle and initialize a communication port transmit and receive data through the port and then close the port if it does not need it any more. The basic program flow is shown in Figure D-4.

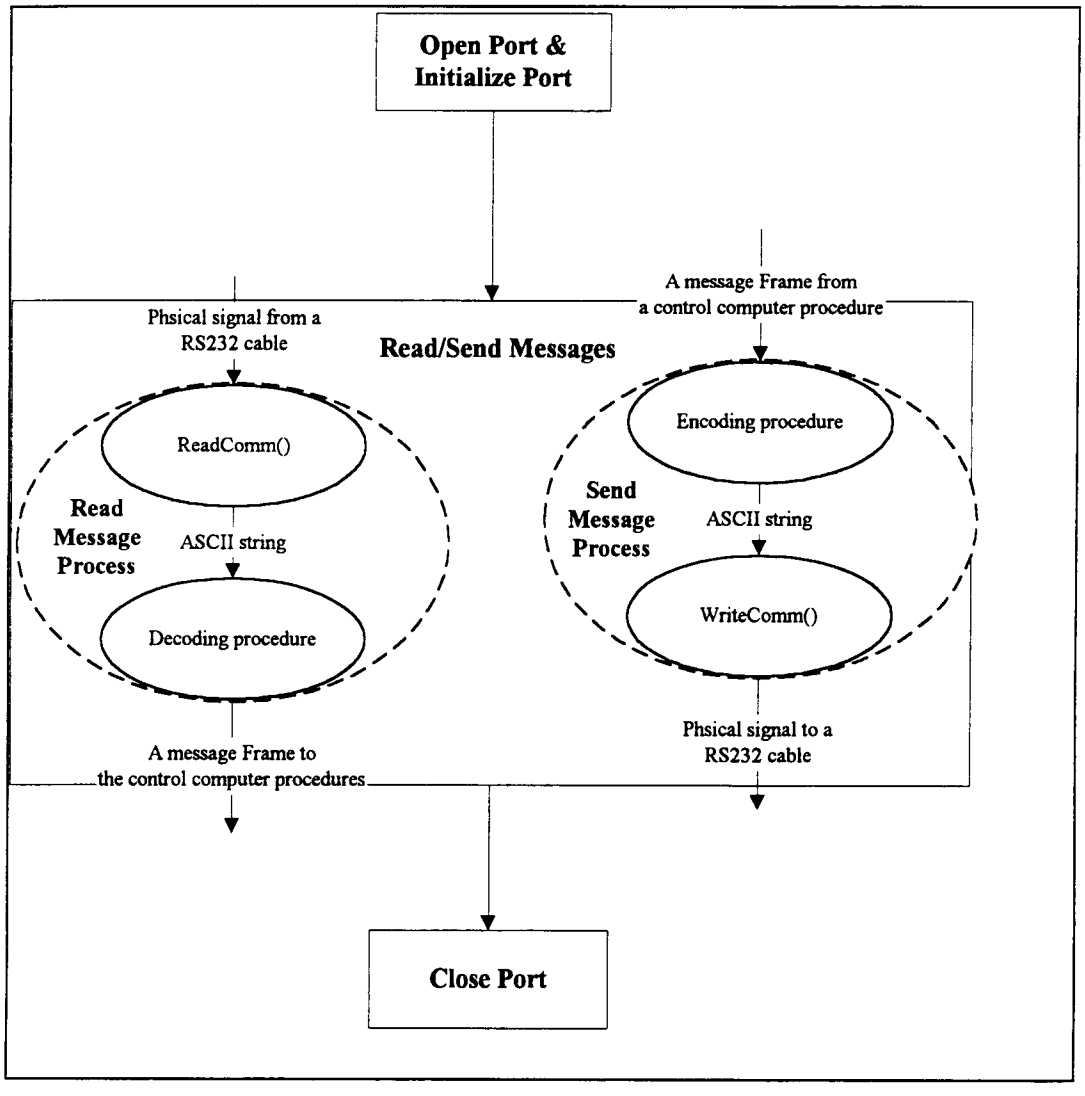


Figure D-4 The RS232 Communication Flow

When a computer needs to send or read a message to/from an outside device, the first procedure is to open a port and initialize it using the OpenComm() function. This routine encapsulates several Windows API calls. It opens the port and sets the port's initial communication parameters. The arguments that this routine accepts include the port name, the queue sizes for the serial port and the initial setting. In this research, the control computer sets up the serial comm. port as "comm. 1", baud rate as " 9600", parity as "even", data bit as "7", stop as "1", input queue size as 2048 characters and output queue size as 2400 characters.

When the control computer needs to receive a message from a remote device via RS232 cable, the control computer initially runs ReadComm() function to translate electrical signals to the ASCII string. Then the decoding procedure decodes this ASCII string into a message frame. A message frame with either probing information or a command is used in other functions, such as SPCtests() or Procedure(). The procedure for sending a message to the destination device via RS232 cable (as shown in Figure D-4) is the reverse procedure for reading of messages.

Finally, when the control computer does not need RS232's reading and sending functions, it must close the communication port so other applications can use it. (For more details in RS232 serial communication in Windows see Monk [1993].)

D-4 The Communication Testing Interface

A testing interface (Figure D-5) is designed to test the communication function and the vise's control (open and close). Its testing functions are as follows:

- Send Byte: It checks the data link layer sending function, Send(to, command). Users can input the integer numbers of "To" and "Command" in their edit boxes. When the "Send Byte" button is clicked , the message is sent to the I/O bus. For example, when users want to send the signal "Next" (value 7) to the conveyor (value 7), the users input number 7 for both boxes (as shown in Figure D-5). If the "Off" button is clicked, the sending bus output will be reset to (0, 0), and the signal will be turned off. Finally, the users use the mouse of the control computer to click on the "Send Byte" button to send the "Next" command to the conveyor via the I/O bus.
- Receive Byte: It checks the data link layer I/O receiving bus's input Wait(to, command) when the "Receive Byte" button is clicked. The received electrical signal is decoded and shown in "From" and "Command" boxes. For example, in Figure D-5 the control computer received 7 in the "From" box and 5 in the "Command" box from the I/O bus implying that the control computer received a "Ack" signal from the control computer.

- To MC: It will send the signal which is highlighted in the scroll list box when the "To M.C." button is clicked. For example, in Figure D-5 the "Start/Probe"signal is chosen. When the "To M.C." button is clicked , the serial plus signal followed by Delimit->Long signal->Short signal->Delimit is sent to the machining center. The machining center will decode this signal to the integer number 6; it means that start a manufacturing cycle with probing procedure for the finished part.

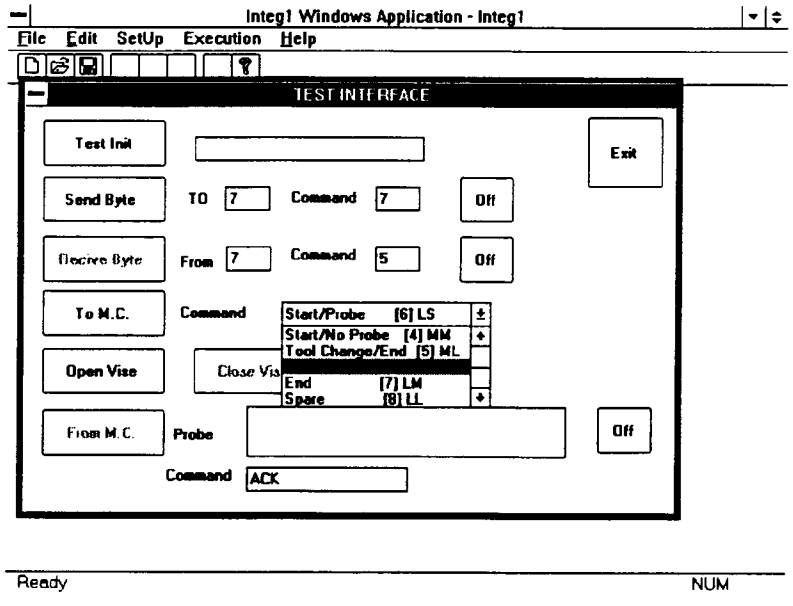


Figure D-5 The Test Interface Screen Display

- Open Vise and Close Vise: Users can open or close the vise in the machining center.
- From M.C.: It is used to test the message received from the machining center via the RS232 cable. When a user clicks "From M.C." the RS232 protocol starts. This protocol will open and initialize the serial comm. port and set up transmission parameters along with receiving the probing data and command message from the M.C. The control computer will decode the ASCII string from the MC to construct the received string as a frame. If the received message is a command-type string, the message's Type is set to Command and its command will be shown in Command's box. If the message is probing data, the decoded message will be shown in Probe's box. For example, in Figure D-5, the M.C. sent an "Ack" command message to the control computer. After processing by RS232

protocol ReadMsg() function, the electrical signal has been interpreted as "Command: Ack" which is identical with the sent ASCII string in the machining center. A decoded function is used to decode this string as the message frame. An example is shown in Table D-5.

Table D-5 The Message Frame Data Structure (The Example Of Receiving An Ack)

Parameter Name	Data Type	Value	Comments
Type	Message Type	COMMAND	Type can be either COMMAND or PROBE
Command	Command Type	Ack	When the message type is PROBE, the Command is "NONE".
Offset	integer	0	When the message is Message type, its Command, Offset, Nominal, and Deviation are set to 0.
Nominal	float	0	
Deviation	float	0	
received string	char	" COMMAND: ACK "	Copy the received string from the machining center.

APPENDIX E DATA FOR REAL-TIME SPC TESTING

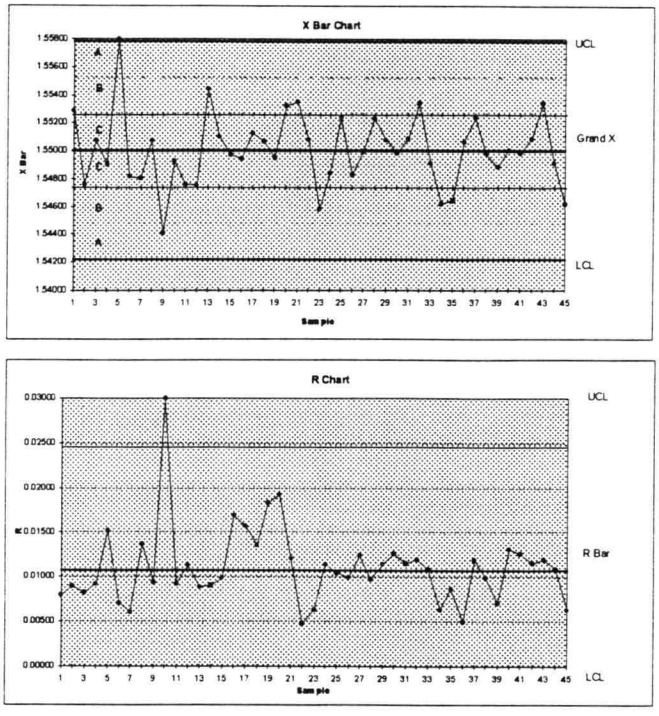


Figure E-1 Extreme Point Test (For Both X bar and R Chart)

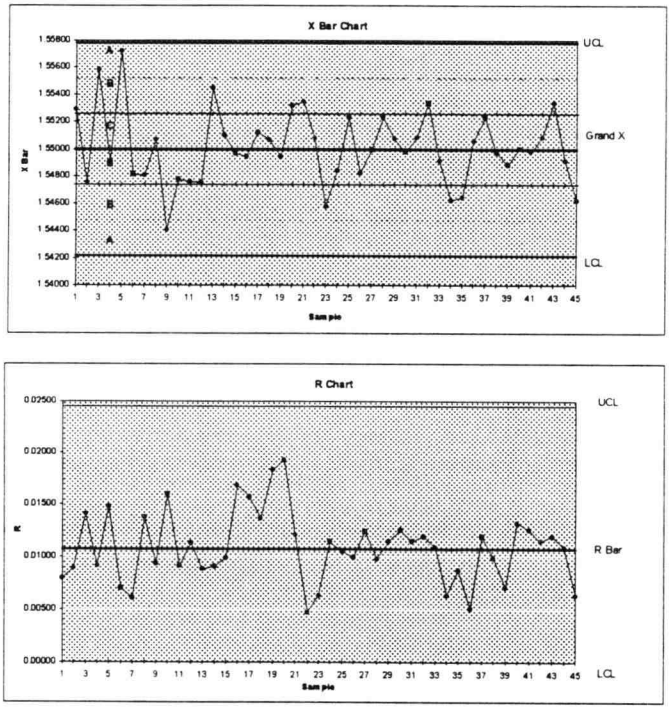


Figure E-2 Two Out Three Points In Zone A Or Beyond (X bar Chart)

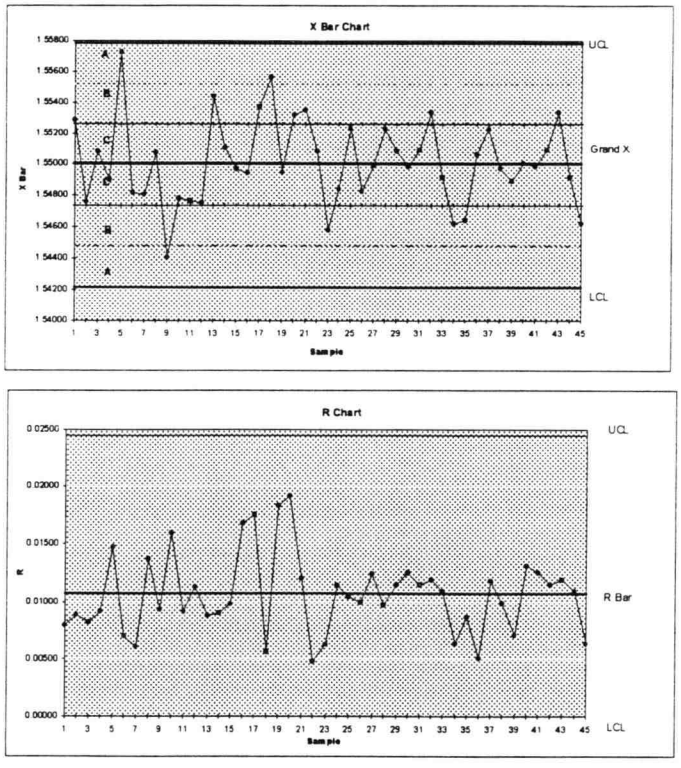


Figure E-3 Four Out Of Five Points In Zone B Or Beyond (X bar Chart)

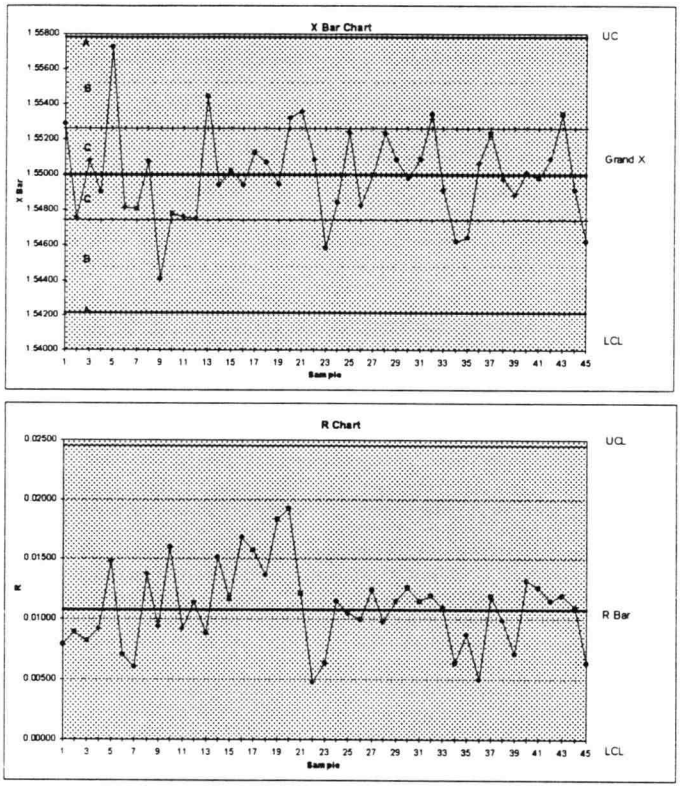


Figure E-4a Run Above Or Below The Center Line (R Chart)

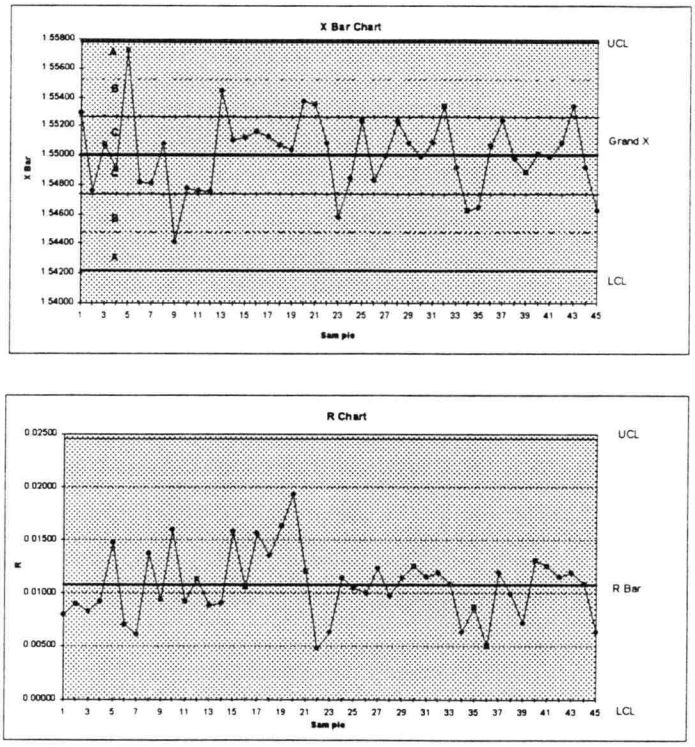


Figure E-4b Run Above Or Below The Center Line (X bar Chart)

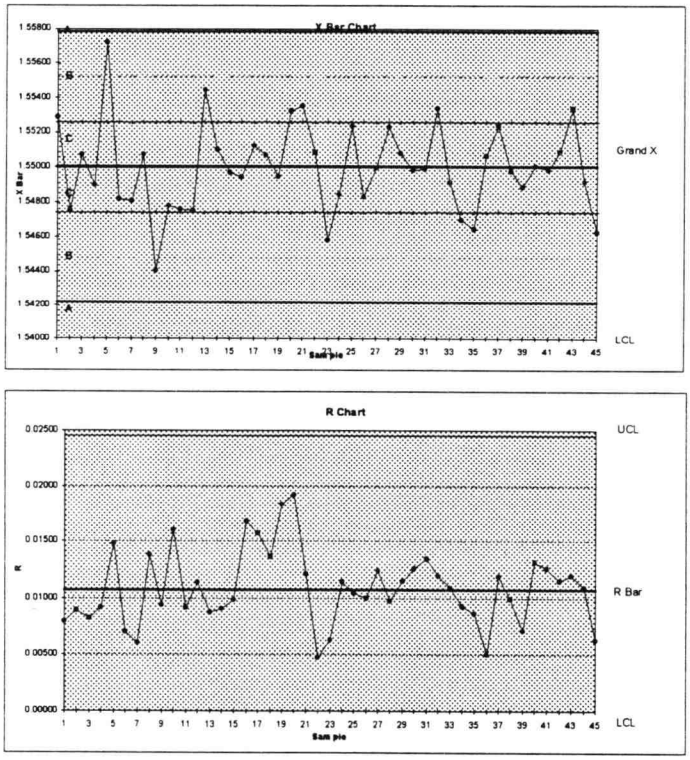


Figure E-5a Linear Trend Identification (R Chart)

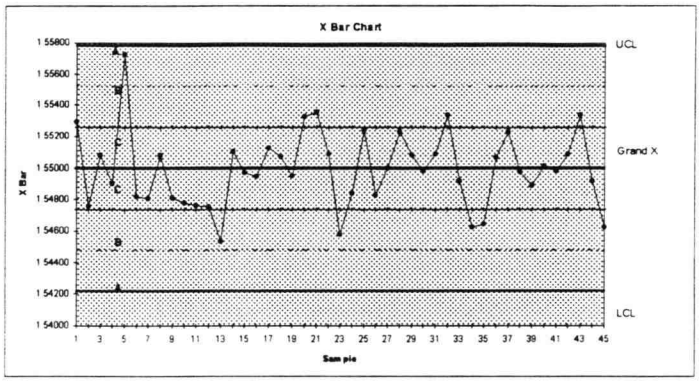


Figure E-5b Linear Trend Identification (X bar Chart)

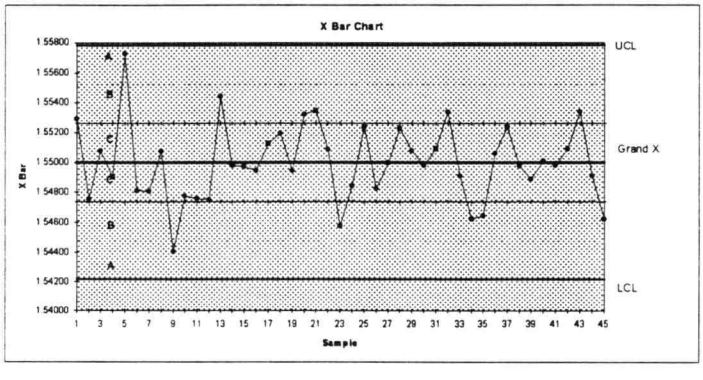


Figure E- 6a Oscillatory Trend Identification (R Chart)

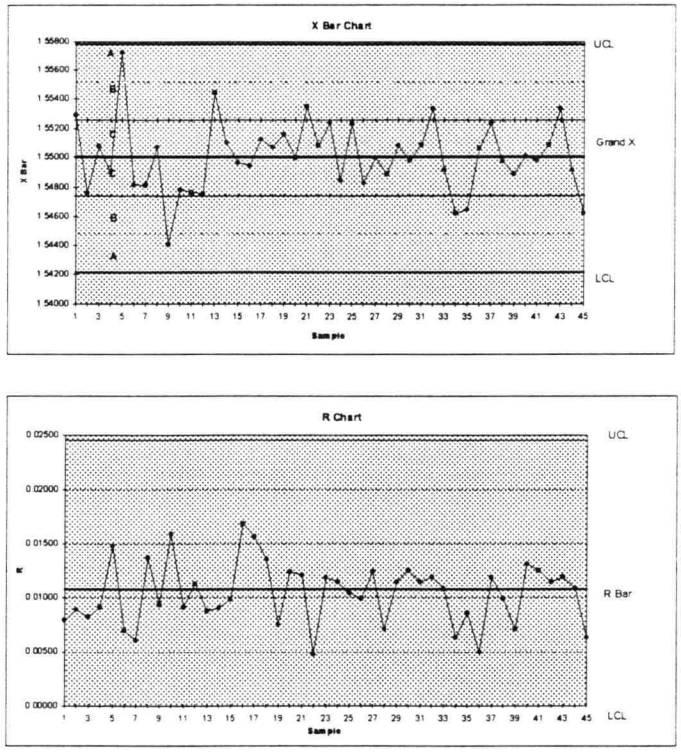


Figure E-6b Oscillatory Trend Identification (X bar Chart)

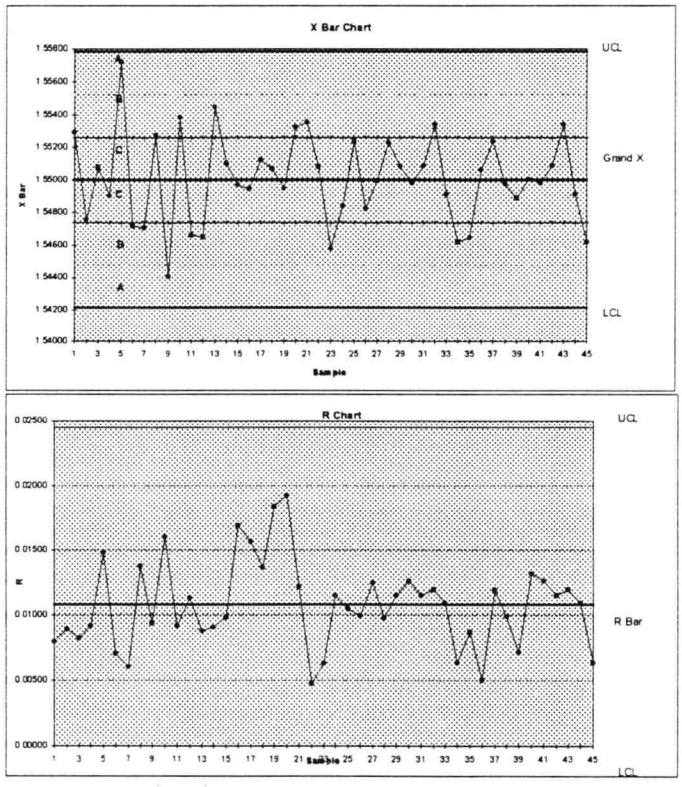


Figure E-7 Avoidance Of Zone C Test (X bar Chart)

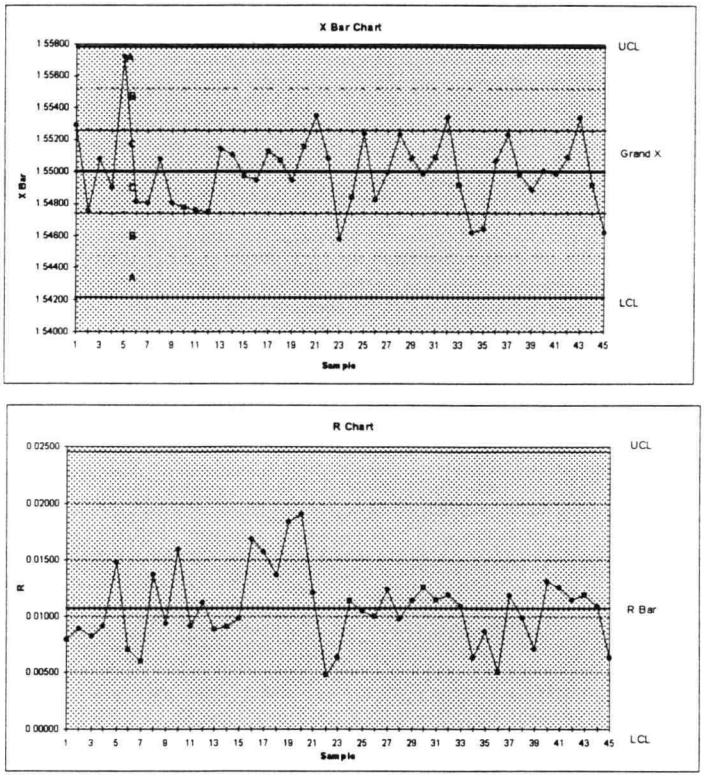


Figure E-8 Run In Zone C Test (X bar Chart)

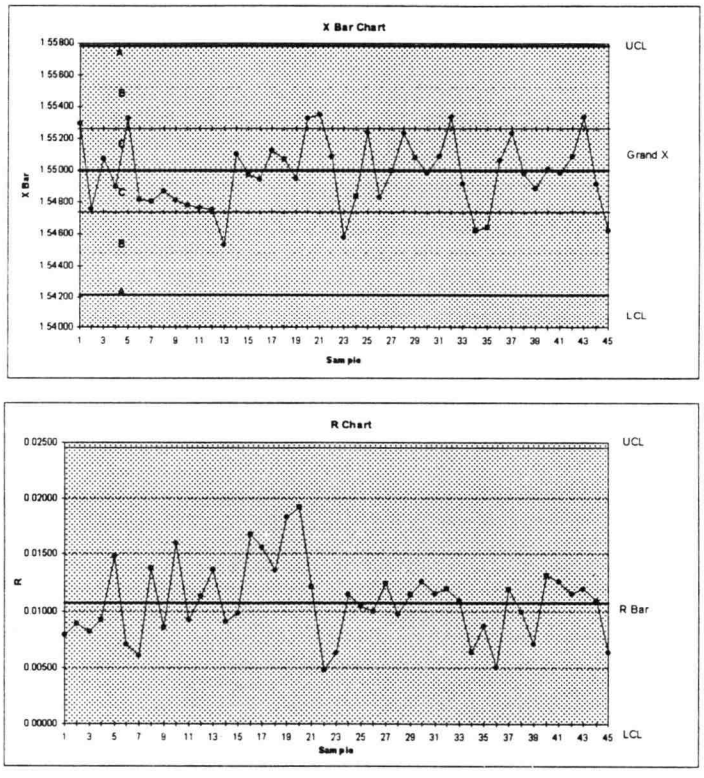


Figure E-9 Tool Wear Out Test (X bar Chart)