COMMUNICATIONS WITHIN A COMPUTER INTEGRATED MANUFACTURING ENVIRONMENT

by

Girish Nair

Project Report submitted to the Faculty of the
Virginia Polytechnic Institute and State University,
in partial fulfillment of the requirements for the degree
Master of Engineering
in
Industrial Engineering and Operations Research

APPROVED:

Dr. M. P. Deisenroth (Chairman)

Dr. E. C. DeMeter

Dr. R. J. Reasor

June 1990
Blacksburg, Virginia.

COMMUNICATIONS WITHIN A COMPUTER INTEGRATED MANUFACTURING ENVIRONMENT

by

Girish Nair

Dr. Michael P. Deisenroth, Committee Chairman

Industrial and Systems Engineering

(ABSTRACT)

In a computer integrated manufacturing (CIM) environment, efficient data exchange and real-time error recovery are required in order to provide a flexible and reliable system. Adoption of a distributed processing network, with several locally intelligent devices, helps satisfy these requirements of CIM. Distributed processing necessitates that computers have the ability to communicate with the following: users, intelligent machines and devices, and with other computers themselves.

In a flexible manufacturing system (FMS) involving processing and interactions between several devices, communication problems often occur. This project is directed at providing test and debug capability for the

various devices within the FMS described below. Additionally, it provides a demonstration of the various interactions between the devices, both in a primitive and an integrated state. For the primitive state, software tools to test the functional status were developed. These tools make the lower level communications transparent to the users, who need be concerned only with their operation.

Briefly, the FMS under consideration is comprised of three function based workcells - for the machining, assembly and material handling functions of the system. Each workcell has under its domain equipment needed to achieve its respective function. The devices in the system include IBM robots, DYNA CNC machines, a conveyor, a programmable logic controller (PLC), and other cell controllers. The FMS is to be used as a demonstration vehicle for students, to teach fundamental principles of system integration and control.

ACKNOWLEDGEMENTS

This work is dedicated to my parents in gratitude for their constant love, support and encouragement. I have achieved everything I have through their blessings.

To Dr. M. P. Deisenroth, my advisor, thank you for introducing me to the world of automation. It has been a learning experience working with you.

I would like to thank Dr. R.J. Reasor and Dr. E.C. DeMeter for serving on my committee.

I would also like to thank my sisters and brothers - in-law for their constant moral support and encouragement.

I would also like to thank my friend C.K. Muralikrishnan, for all his suggestions. Finally, to my friend and roommate Vishwanath Tirupattur for being there, through high and low. I can't thank you enough.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Better productivity is an increasingly important requisite to remain competitive in the worldwide market of the manufacturing industry. Factory automation and computer integrated manufacturing (CIM) are two strategies that can be applied to give a company better productivity and competitiveness. CIM is a technology that involves the integration of the product design engineering, manufacturing and management systems into a synergistic system, where the whole is more productive than the sum of the parts. However, CIM systems must be carefully planned to succeed. Automating operations should be undertaken in phases, implementing computers for a few functions at a time.

Adoption of a "top-down-bottom-up" implementation philosophy has been successful for most CIM systems. This philosophy advocates planning from the "top-down" by strategists, who have an entire picture of the system they wish to integrate. The "bottom-up" part of the philosophy requires implementation of the plan starting at the lowest level and working upwards. The bottom level, namely the shop floor, constitutes the foundation of any CIM effort and

is the level at which any implementation should begin. When operational, the next hierarchical level in the organization is addressed. This upward and outward implementation methodology, can be done in a step by step fashion, with system integration being achieved as the modules are automated.

A flexible manufacturing system (FMS) can be defined as a group of automated machine tools that are interconnected by means of a material handling and storage system, and which operates as an integrated system under computer control [18]. Most manufacturers define an FMS cell (or workcell) as the smallest building block of an FMS. An FMS cell has under its domain one or more devices, like robots, CNC machines, inspection and/or material handling equipment. The devices themselves are under the control of a reprogrammable workcell controller (WCC). The WCCs themselves can be under the control of higher level supervisory controllers.

There are many factors to be considered during CIM implementation. During the design of an FMS within a CIM environment, the software control strategy, the physical and electrical configurations have to be decided. Floor space constraints, the work envelopes of the different devices,

the maximum allowable length of cables for various communications, the minimization and ease of material handling, operator interface and safety considerations are factors influencing the physical configuration and layout of the system.

2. PROBLEM STATEMENT AND OBJECTIVES

A flexible manufacturing system (FMS) is normally divided into smaller workcells, each of which consist of a variety of equipment built by different manufacturers. Since vendors of automation equipment have no standard code to which they have to adhere, integrating these machines under a flexible manufacturing environment becomes a problem [9]. An FMS requires the integration and coordination of a diverse group of machines, each operating on a different protocol and configuration. In systems which involve a large number of interfaces between devices and require synchronization of tasks, it is necessary to have a parallel intelligent device like a programmable logic controller (PLC), interfacing with the devices and the cell controllers.

Another problem found in systems which involve a large number of interfaces between devices, is that the amount and complexity of the communications that are necessary can become overwhelming. In order to have a flexible and reliable system, it is imperative to be able to

test and debug the functional status of the communication links within the system.

There were two objectives of this research. The primary objective was to develop a test and debug capability for the communication links within the FMS under consideration. Software tools to test the functional status of the system were developed, towards meeting this objective. In other words, the tools developed can be used to check the hardwire continuity of the system. The second objective was to demonstrate the applicability of primitive communications to achieve high level operations required for an integrated and automated system.

3. LITERATURE REVIEW

Having a manufacturing edge in the marketplace is primarily determined by the efficiency and quality of the whole manufacturing process. In today's changing market however, flexibility has become an increasingly important requisite for corporations to remain competitive. Incorporating flexible automation and CIM is a solution to these problems.

CIM and Cell Control

CIM has been defined as a method of providing computer assistance, control and high level integrated automation at all levels of the manufacturing industry, by linking "islands of automation" or cells, into a distributed processing system [1]. Each cell can be comprised of one or more devices, like robots, CNC machines, conveyors, vision systems, coordinate measuring machines and/or AGVs, under the control of a cell controller.

One of the fundamental services of a cell controller, is the control and execution of activities of the devices within the cell, in a desired activity sequence. Real time predictability must be maintained by the cell controller, for situations requiring synchronized coordination of device activities. The cell controller can ensure real time synchronization between the devices under its control, by incorporating a predetermined handshake policy.

Current cell controller offerings generally serve as system integrator's toolkits, and as such, are composed primarily of software utilities [14]. However, in order to implement CIM successfully, three types of integration are required - electronic, physical and organizational [5]. Manufacturers have failed to realize that in attempting to tie together and control computerized processes, they need to establish a control hierarchy [7].

Control Hierarchies

In order to achieve flexibility and modularity in an advanced manufacturing system, a well defined control strategy is necessary. There are many control strategies, the most popular of which are the hierarchical control model developed by the National Bureau of Standards (NBS) for

their Automated Manufacturing Research Facility (AMRF), and the DEC/Philips control system model.

In order to meet the requirements of real time production, the NBS hierarchical control system [14], has been:

1) partitioned into a hierarchy in which the control processes are isolated by function and communicate via standard interfaces,
2) designed to respond in real time, to performance data, derived from machines equipped with sensors,
3) implemented in a distributed computer environment, using recent advances in software and hardware engineering.

The NBS system architecture is a command/feedback control structure composed of five major levels: facility, shop, cell, workstation and equipment (Figure 3.1). This configuration ensures that the size, functionality and complexity of individual control modules are limited.

The DEC/Philips model of a controller [5], on the other hand, has access to local and global data and commands from the controller above it. The model issues commands to the subordinate controllers and receives sensory input from

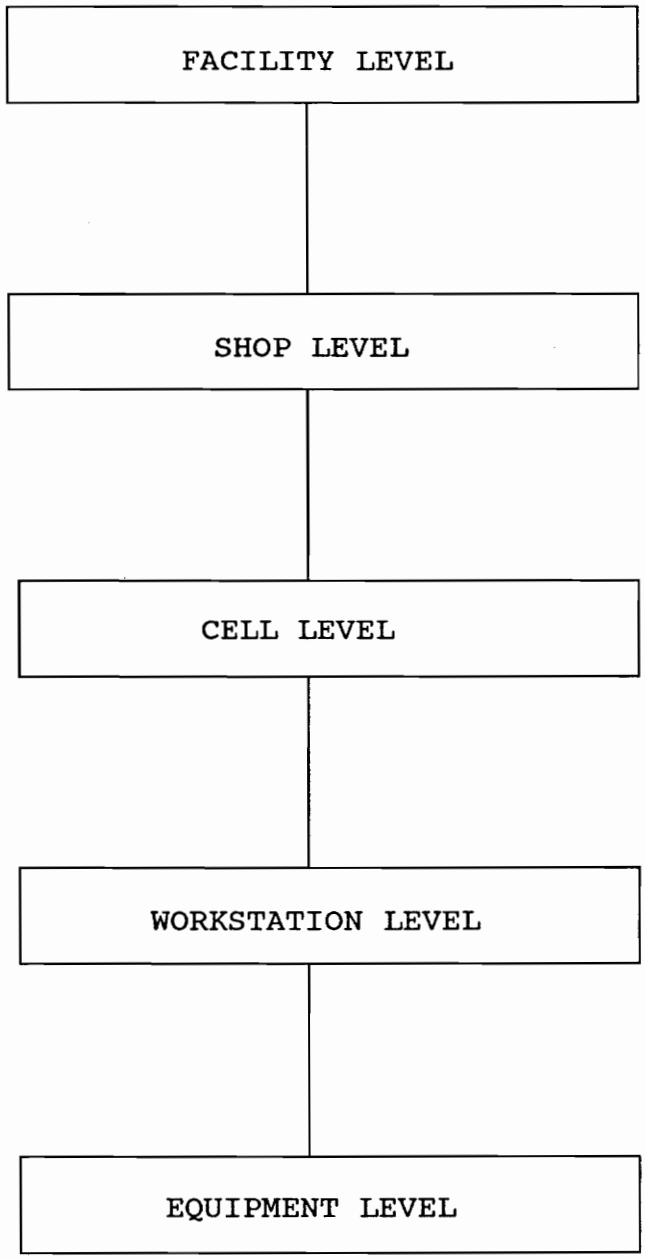


Figure 3.1 NBS control hierarchy

them. The three major functions performed are task decomposition, sensory data processing, representation, storage and retrieval.

Previous Relevant Work at VPI&SU

As mentioned previously, the development of a CIM system can be executed in stages, implementing a few functions at a time. This is the manner in which the development and implementation of the FMS in the CIM facility at VPI&SU has been undertaken. Previous work includes:

1) Development of a basic set of macros, by Guleri [9], which permit communications between the workcell controller and the robots and the CNC machines. Primitive routines to receive and send data from serial ports, and macros to download to the CNC machine was developed by Guleri. The macros developed for the robot included the ability to send the robot "home", set the robot controller to the "auto" mode, download files, select partitions and start and stop the robot cycle.
2) Development of software menu system/workcell controller programs in graduate class group projects, namely Saboo et al. (Systems Control) [21], Romano et

al. (Machining) [20], Nair et al. (Assembly) [16] and Ridgway et al. (Material Handling) [19].

3) Development of a Relay ladder logic (RLL) control program to drive the material handling workcell, developed by Bidani [1] and modified by Muralikrishnan et al. [15]. This program executes different conveyor tasks, depending on the task code it receives from the workcell controller.
4) Development of a simulated intelligent shopfloor control system by Zhang [23].

The first phase of this project, namely building a test and debug capability for the FMS under consideration, used the toolkit developed by Guleri [9] as a basis. The software written for the three workcells and the system controller [15,16,19,20,21] was used as a basis for the development of the integrated FMS software.

4. SYSTEM DESCRIPTION

The CIM laboratory at the Department of Industrial and Systems Engineering at the Virginia Polytechnic Institute and State University, houses an FMS which serves to provide instruction and research facilities in computer control and integration of manufacturing systems. The entities within the laboratory (Figure 4.1) are two IBM robots, two three-axis CNC milling machines, a conveyor system, an automatic storage and retrieval system (AS/RS), a TI 565 programmable controller, a GE Optomation vision system, fixtures, sensors, actuators, pallets, a vibratory bowl feeder, and four AT&T personal computers, interconnected by a STARLAN local network. STARLAN uses a star configuration in which all computers communicate directly with the central computer or file server, which is the fifth computer in the network. The computers may also be "daisy chained" to achieve communications, as shown in Figure 4.2. This is how the computers have been linked to each other in the CIM laboratory. The network has a 1 megabit per second data transmission speed. Note that the server is not to be confused with the systems controller

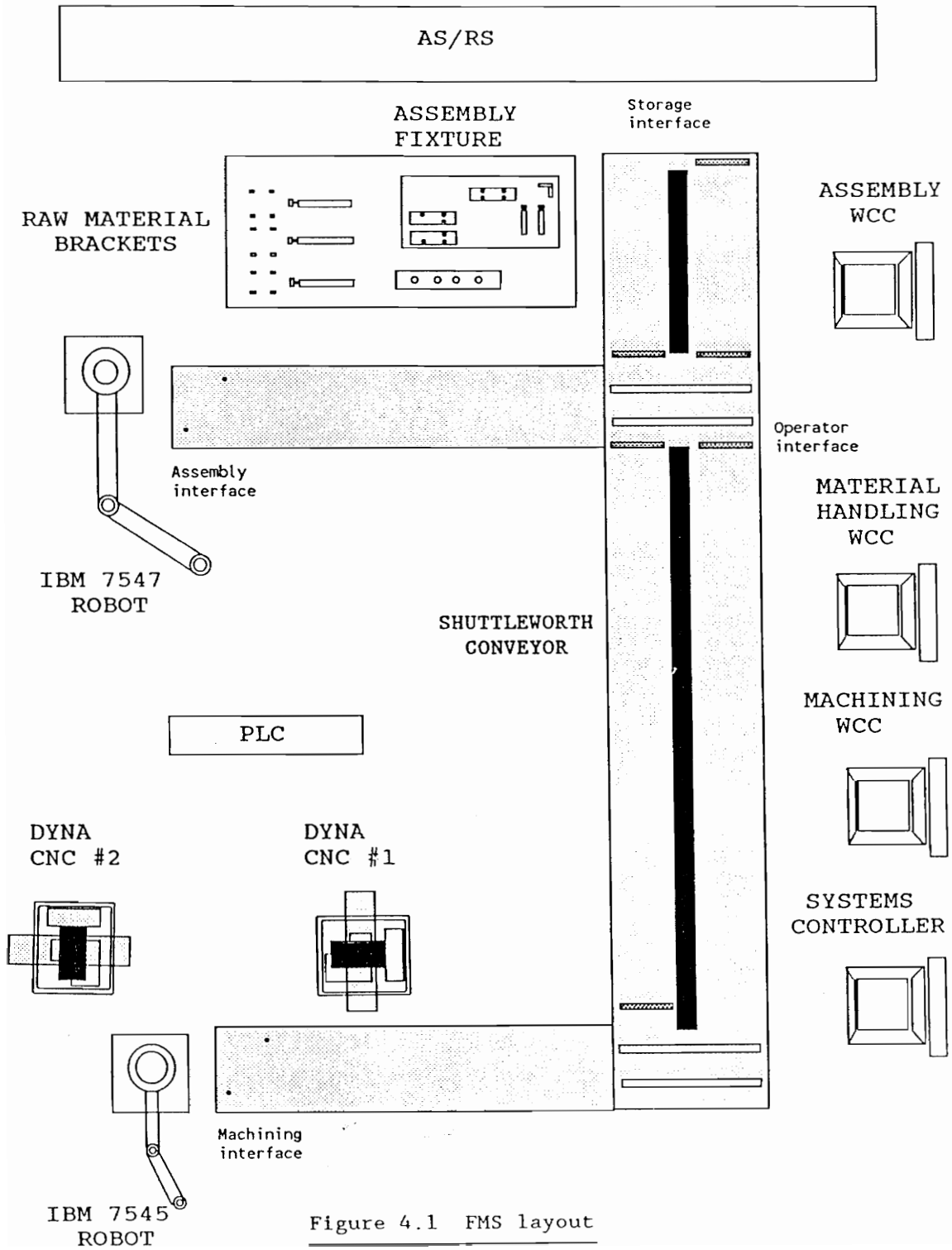


Figure 4.1 FMS layout

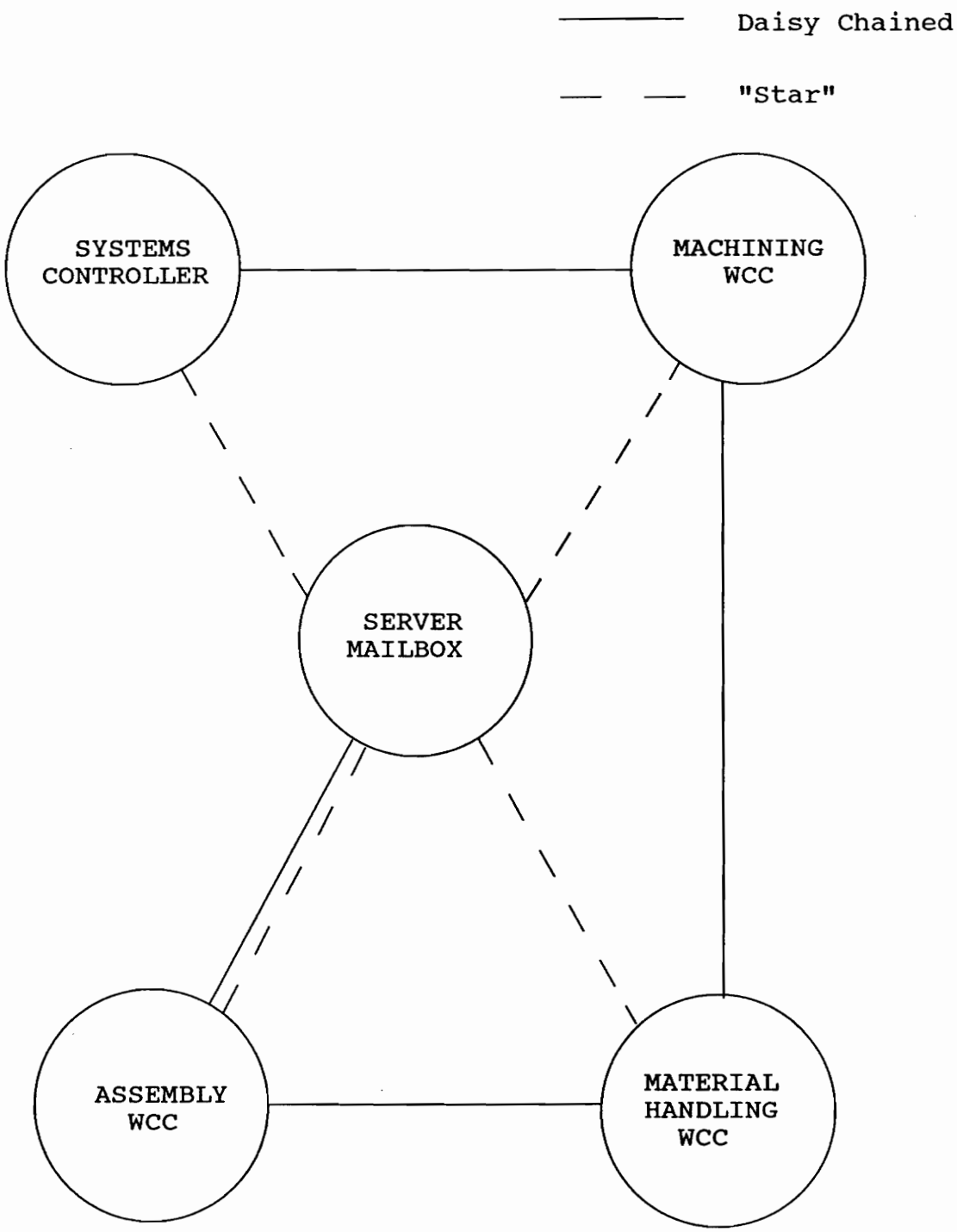


Figure 4.2 LAN configuration

(Figure 4.2). Each computer reads from and writes messages to a common directory or "mailbox" on the server.

The system was originally designed for discrete part batch manufacture of two products - miniature wax SCARA robots and milling machines. A third product, a keychain, was added this academic year. Different products can be manufactured by changing the fixtures appropriately and altering the control sequence software. Final redesign of the fixtures was done by Wiegmann et al. [22] for the machining workcell, and Economy et al. [6] for the assembly workcell.

Conceptually, the development and implementation of the lowest two layers, and part of the third layer of the NBS control hierarchy, namely, the equipment, the cell and the system levels, defines the scope of this project. The systems controller provides overall control of the system. It basically acts as a commander and coordinator of operations at the cell level. Ideally, a systems controller consists of a task manager and a resource manager module [13]. The task manager is responsible for planning capacity, grouping orders into batches, assigning and releasing batch jobs to cells, allocating resources to cells, tracking individual orders to completion, and report

generation. The resource manager on the other hand, allocates production resources according to the optimization algorithm chosen. Only the task manager will be implemented by this project, without any elaborate scheduling features. The tasks will be performed in a pre-determined sequence - the capability to prioritize tasks is beyond the scope of this project.

There are three cell controllers - for the machining, assembly and material handling cells respectively. A detailed description of the equipment and hardware configurations in the workcells as well as software routines developed for the workcell controllers and the systems controller are given in later chapters. The software routines call primitives developed as part of the toolkit.

Error conditions in the case of all three workcells are indicated by digital outputs readable to the programmable controller. This helps in status monitoring and debugging of the system. The CIM laboratory cable documentation [2] gives complete information of the cabling configuration in the present system. Figure 4.3 gives the FMS hardwiring diagram, and Figure 4.4 depicts the communications network in the CIM laboratory.

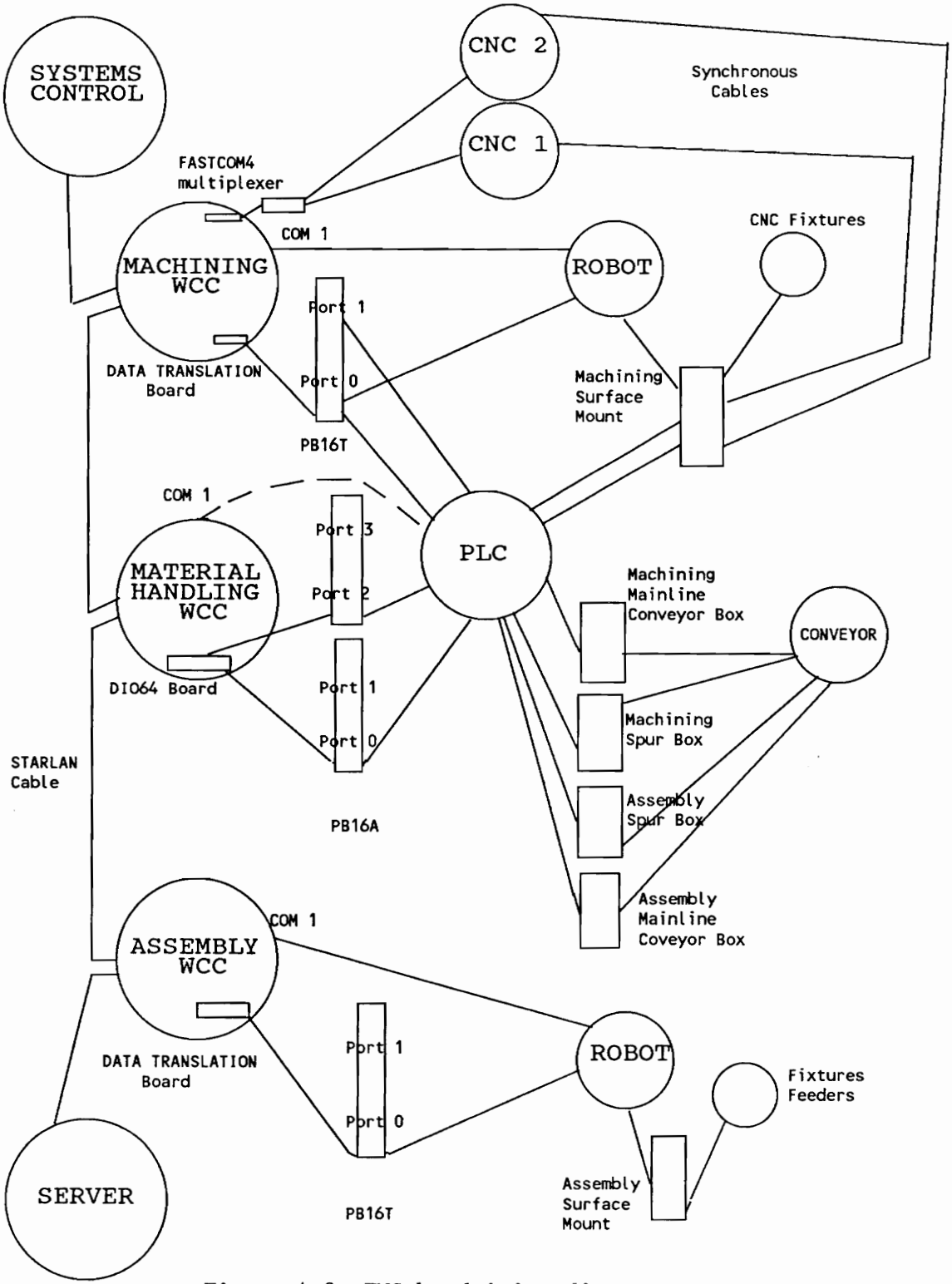


Figure 4.3 FMS hardwiring diagram

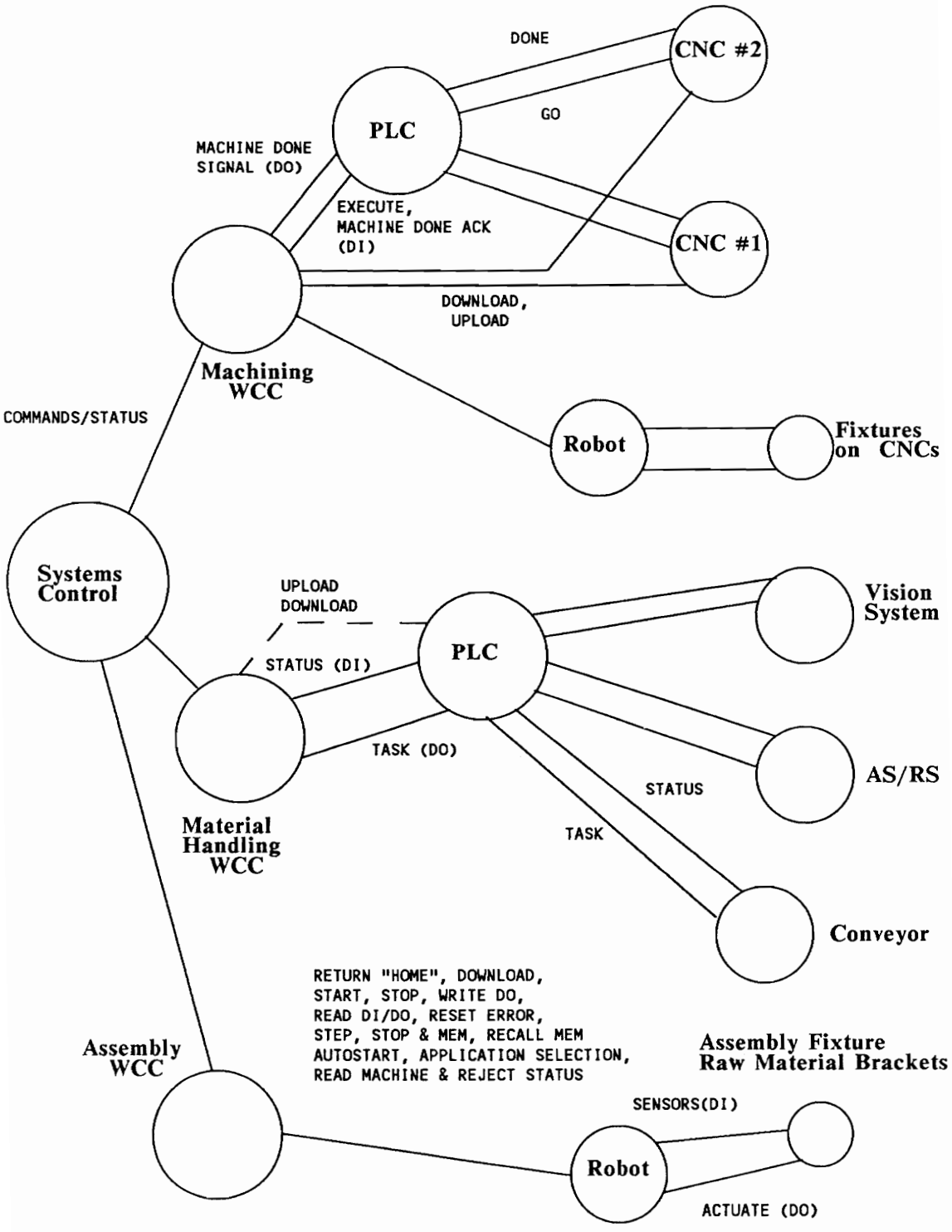


Figure 4.4 Communications in the CIM laboratory

FMS Operations

To illustrate the operations that must take place within the FMS, one can consider the full cycle involving the manufacture of a robot or a CNC machine. An empty pallet is picked up from storage and transported to the assembly workcell interface, where the IBM 7547 robot builds an appropriate raw material kit. The raw material pallet is then taken to the machining workcell, where the IBM 7545 robot loads the parts onto fixtures on the CNC machines. After machining, the finished parts pallet is transported back to the assembly workcell, where the product is assembled by the IBM 7547 robot using fixtures. Finally, the pallet with the finished product is sent back to storage. When the raw material in the feeder brackets in the assembly workcell is in short supply, they must be replenished in order to be able to manufacture more products.

The pallets can be sent to the storage system in many different configurations - empty, stocked with raw material, having robot/CNC kits, or having final robot/CNC assemblies. In the existing setup, since neither the AS/RS, nor the vision system have been incorporated, it is assumed that appropriate pallets are being retrieved at the storage interface of the conveyor (Figure 4.1).

5. METHODOLOGY

The objective of this research was twofold. Firstly, to develop software interfaces for testing and debugging communications between the various devices in the FMS: the robots, CNC machines and the programmable controller. Secondly, to use the software tools or "macros" developed to achieve total integration of the FMS. The method of approach to be followed in attaining this goal was to divide the project into two phases. The first phase was the development of the complete "toolkit" of software modules required to achieve the communications desired between the devices. This part of the project was built on the work started by Guleri [9]. The second phase was to integrate the communication network with interlocking, handshaking and parallel processing features, so that the entire system can function as one unit. This part of the project utilized cell control software developed by Saboo et al. [21], Romano et al. [20], Nair et al. [16], and Ridgway et al. [19].

The development and implementation was done under the DOS 3.3 operating system on AT&T 6300 series computers, in the "C" programming language, using Microsoft C Version 5.1.

A relay ladder logic program, developed by Muralikrishnan et al. [15] on a TI 560/565 PLC for the material handling workcell, was augmented to include sequence control of the machining workcell.

This project has identified five different types of communications necessary to achieve integration of the CIM facility described above. These communications are:

1) Workcell controller <----> Robot
2) Workcell controller <----> CNC machine
3) Workcell controller <----> PLC
4) PLC <----> CNC machine
5) Workcell controller <----> Systems controller

The first phase in the effort was to develop routines to provide these communications. These software tools permit the future user to use the communication routines without requiring an in-depth knowledge of the low-level communications involved. The second phase was to integrate all the communications necessary to operate the system as one unit (Figure 4.4), sequentially in a desired manner. The implementation of the second phase was made feasible by using the tools developed in the first phase of the project. Specifically, the tasks involved in both phases of the project are identified in Table 5.1 and Table 5.2.

TABLE 5.1

Test and Debug Capabilities (Augmenting Toolkit)

1) Controller to Controller LAN Interface	a) Read file in linked directory
	b) Write message to file in linked directory
2) Controller to Controller Serial Communications	a) Read
	b) Write (1) File (2) Keyboard
3) Workcell Interface with Robot	a) Execution of Task Code Generated from Keyboard
	b) Error Resetting
	c) Reading Digital I/O
	d) Writing Digital Outputs
	e) Read Machine Status
	f) Read Reject Status
	g) Step mode
	h) Stop and Mem
	i) Recall Memory
	j) Unload a partition
4) Workcell Interface with CNC	a) Upload file from CNC
	b) Execution of commands generated at keyboard

TABLE 5.1 continued

5) Workcell Digital I/O and Interface with PLC	a) Ability to read inputs and write outputs with DATA TRANSLATION 2808 and DIO64 boards
6) PLC Interface with CNC(s)	a) Ability to have "handshake"

TABLE 5.2

System capabilities

1) Control of primitive functions of the devices in the workcells "manually" by the cell controllers.
   Example: Read I/O, Write outputs, Download, Error Reset, Read/Write from/to I/O Boards, etc.
2) Control of high level cell tasks by the respective cell controllers.
   Example: 'Mirobot', 'Asrobot', 'Sttoop'
   (refer Tables 7.4, 8.2, 9.1)
3) Control and status monitoring of the cell controllers by the systems controller.
   Example: Any macro in 2) above.
4) Control of the primitives of any device in any workcell by the systems controller.
   Example: Any primitive in 1) above, except "read" functions of robots of machining and assembly workcells.
5) Report generation of the tasks performed in a session by the systems controller.

6. TOOLKIT FUNCTIONS

The toolkit functions developed as the primary objective of this project can be divided into the following categories:

1) LAN communications routines
2) Serial communications routines
3) Robot primitives
4) CNC primitives
5) Digital I/O routines for DATA TRANSLATION 2808 board
6) Digital I/O routines for DIO64 board

In order to enable portability, the routines for all of the above are in "include" files, which are named according to the type of routines they contain (Table 6.1). A future user can directly use any of the subroutines in the toolkit, by including the relevant "include" file in a calling program.

An executable **"toolkit"** program was created to permit operator interaction with the various workcell entities through keyboard entries. This will permit future system users to check the functionality of various toolkit

TABLE 6.1

Toolkit functions specifications

Routines marked with a * are modified routines developed by others. Routines marked with a ** are unmodified routines developed by others. Unmarked routines have been independently developed.

Include File		Function
tools/lan.h	**	int communicatein (char *flname, char *mesg)
	**	int communicateout(char *flname, char *mesg)
		int c2ommunicatein (char *flname, int *sub2, char *sub3, char *mesg)
		int c2ommunicateout (char *flname, int *sub2, char *sub3, char *mesg)
tools/serial.h	*	unsigned initcom(int port, unsigned comdata)
	*	unsigned getcomstatus(int port, unsigned comstatus)
	*	unsigned sendcom(int port, unsigned char com_char)
	*	unsigned receivecom(int port, unsigned char com_char, unsigned *com_error)
		int readserial(int port, unsigned comdata)
	*	int sendfile(int port, unsigned comdata, char *flname)
		int keybrdwrite(int port, unsigned comdata)
tools/robot.h	*	int execute(char rc1, char rc2)

TABLE 6.1 continued

	*	int time_delayfn(char rc1, char rc2)
		int readrobot(char rc1, char rc2, int nrec)
	*	int downloadrob(int *partno, char *name)
		int write_do(char donum, char onoff)
		int return_home()
		int autostart()
		int application(int *partno)
		int start_cycle()
		int stop_cycle()
		int error_reset()
		int step()
		int stop_and_mem()
		int recall_mem()
		int read_io()
		int read_reject_status()
		int read_machine_status()
		int unload(int *partno)
tools/cnc.h	**	int download(int port, char *filename)
		int upload(int port, char *filename)
tools/data_translation.h		
		int m10()
		int m11()
		int m12(int indata)

TABLE 6.1 continued

	int m13(int value)
	int dtread()
	int dtwrite(int value)
tools/dio64.h	int d64read()
	int d64write(int port, int value)

programs external to their specific software. The toolkit main menu is shown in Figure 6.1. Subsequent sections of this chapter discuss the functioning and specifications of the routines developed.

LAN communications functions

Routines providing the ability to read from or write to a file in the linked directory of the LAN, form the offerings of this section of the developed toolkit. These routines are in the "include" file 'lan.h'. The routines are useful in message passing between computers (example: between a cell controller and the systems controller).

int communicateout(char *flname, char *mesg)

The communicateout() function enables writing a character string 'mesg' to a file 'flname' stored in the linked directory or "mailbox". If the file 'flname' already exists, it is overwritten.

int c2ommunicateout(char *flname, char *mesg, int *sub2, char *sub3)

The c2ommunicateout() function enables writing a character string 'mesg', a pointer to an integer '&sub2' and

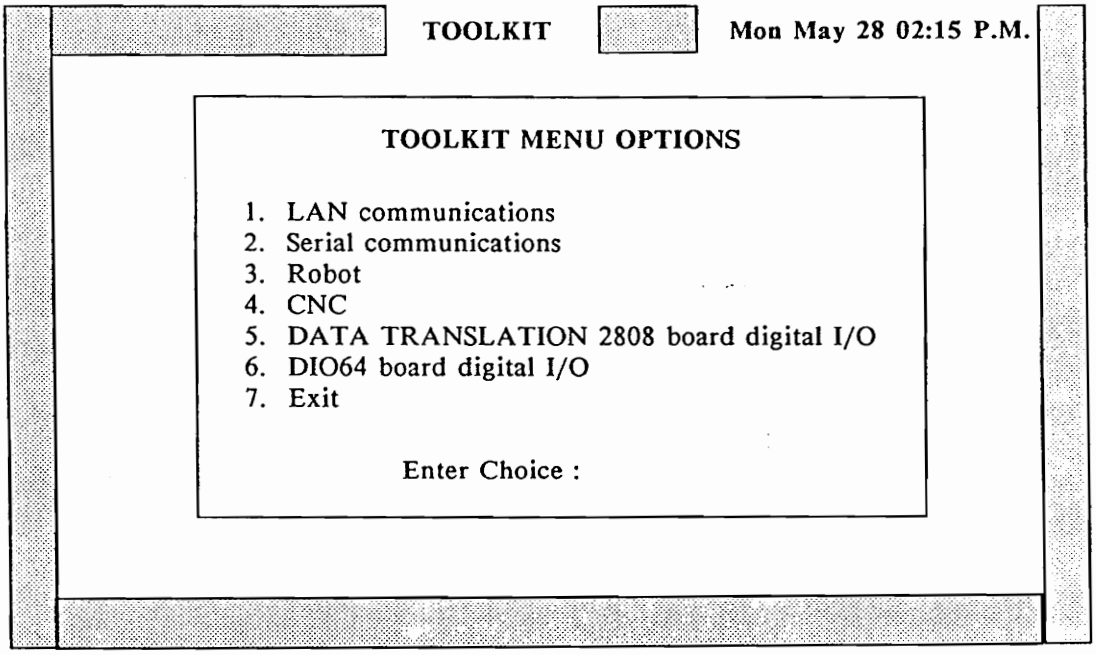


Figure 6.1 Toolkit menu

another character string 'sub3' to a file 'flname' in the linked directory or "mailbox". If the file 'flname' already exists, it is overwritten. This function is especially useful if besides the string message 'mesg', certain parameters also have to be passed between computers.

int communicatein(char *flname, char *mesg)

The communicatein() function enables reading a string message 'mesg' from a file 'flname' in the linked directory or "mailbox". If the file does not exist, the function returns an integer value of 901.

int c2ommunicatein(char *flname, char *mesg, int *sub2, char *sub3)

The c2ommunicatein() function enables reading a character string message 'mesg', a pointer to an integer '&sub2' and another character string 'sub3' from a file 'flname' in the linked directory or "mailbox". If the file does not exist, the function returns an integer value of 902.

The routines communicateout(), and communicatein() are counterpart routines which can be used by two computers to communicate over a LAN. Similarly the routines

c2ommuinateout() and c2ommunicatein() are counterpart routines.

The executable file 'toolkit.exe' (Figure 6.2) exercises the function communicatein() by displaying the string 'mesg' in 'flname'. Communicateout() is exercised by setting the first string input from the keyboard to the variable 'mesg' and then writing this variable into the file 'flname'.

Serial communications functions

The routines developed to enable serial communications between two devices are in the "include" file 'serial.h'. Generic communication routines initcom(port, comdata), getcomstatus(port, &comerror), sendcom(port, send_char) and receivecom(port, &send_char, &comerror) had been developed by Guleri [9] earlier as part of his toolkit. These routines themselves used the standard primitive _bios_serialcom() routines of the Microsoft C 5.1 BIOS library, to perform the basic serial I/O operations of initializing the port, reading the status of the port and sending and receiving characters through the port respectively.

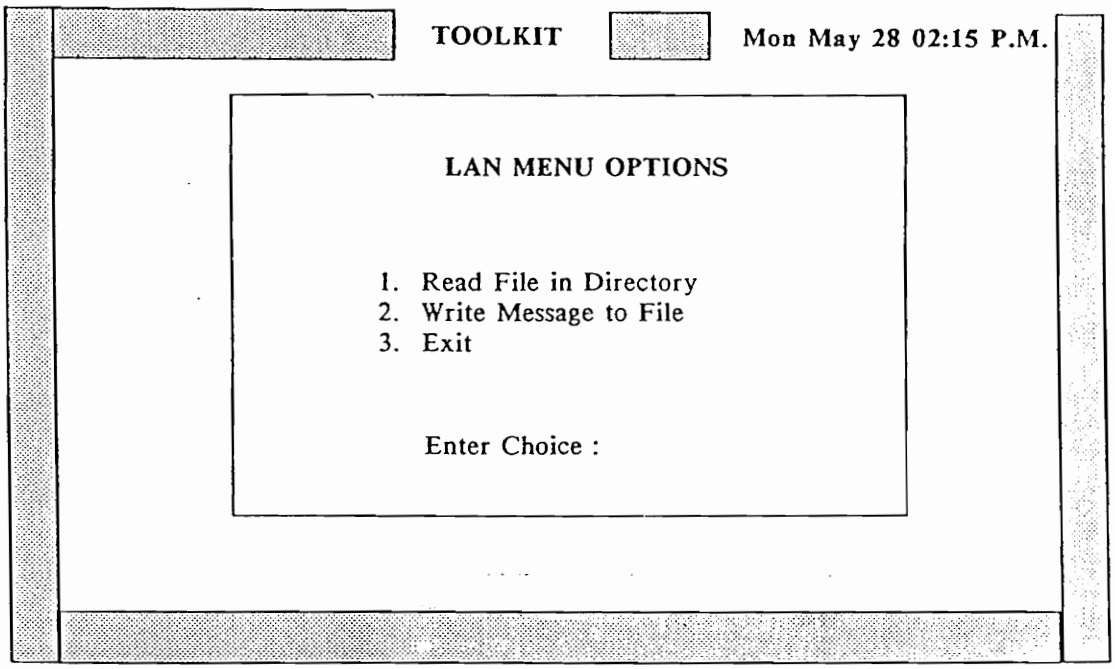


Figure 6.2 LAN communications menu

The above four routines have been used to develop the subroutines read_serial(), sendfile(), and keybrdwrite(), (Figure 6.3) which can be used to communicate with any device using serial communications in a half duplex mode.

int read_serial(int port, unsigned comdata)

The utility read_serial() polls the serial port number 'port' for input. The port number is always 1 less than the COM number of the port. The variable 'comdata' is used to set the parameters of the communication line. It is obtained by ORing the constants which specify the baud rate, parity, word length and number of stop bits for the transmission. The function stores the characters it receives in the global array variable combuffer[900]. The function keeps polling the serial port, till it receives an EOF (end of file - hex 1A) character. An example of where this routine can be used is in the uploading of files from a CNC machine to the host computer.

int sendfile(int port, unsigned comdata, char *flname)

The utility sendfile() sends the file 'flname' character by character to the serial 'port' for output, using the transmission parameters specified by 'comdata'. An example of where this routine can be used is in the downloading of a file to a CNC machine.

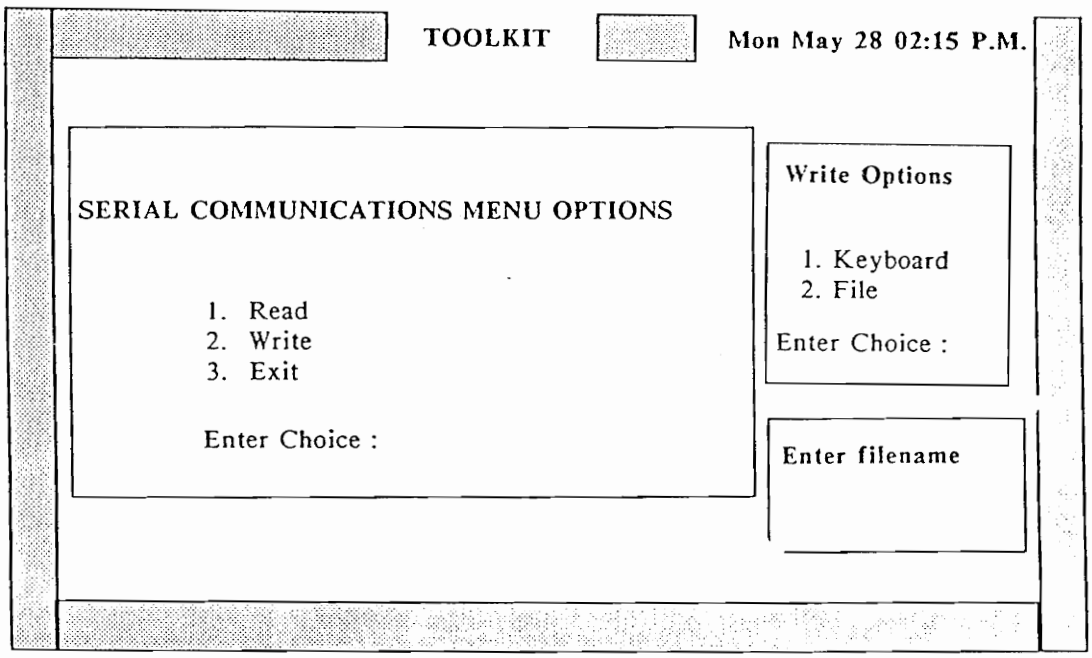


Figure 6.3 Serial communications menu

int keybrdwrite(int port, unsigned comdata)

The routine keybrdwrite() allows the keyboard operator to send characters through the serial 'port', using transmission parameters specified by 'comdata'. This utility can be used to develop, download and execute whole CNC programs on-line. It can also be used to help debugging communications with the robot since it can be used to achieve any operation in the robot library, provided the operator/user keys in the correct sequence of characters, and in accordance to the protocol demanded by the robot controller. To use the above utilities for communications between two intelligent controllers, in which both ends act as DTEs (Data Terminal Equipment), it is necessary to wire the RTS to the CTS at each end (that is, have a "null-modem"). This is the case for a robot - computer and for a computer - computer interface, but not for the computer - CNC interface (Appendix B). The routines described above are powerful tools for debugging communication lines.

The routines can be exercised in the "toolkit" program, by first setting the transmission parameter variable 'comdata' using the menu shown in Figure 6.4, and then choosing any of the options shown in Figure 6.3.

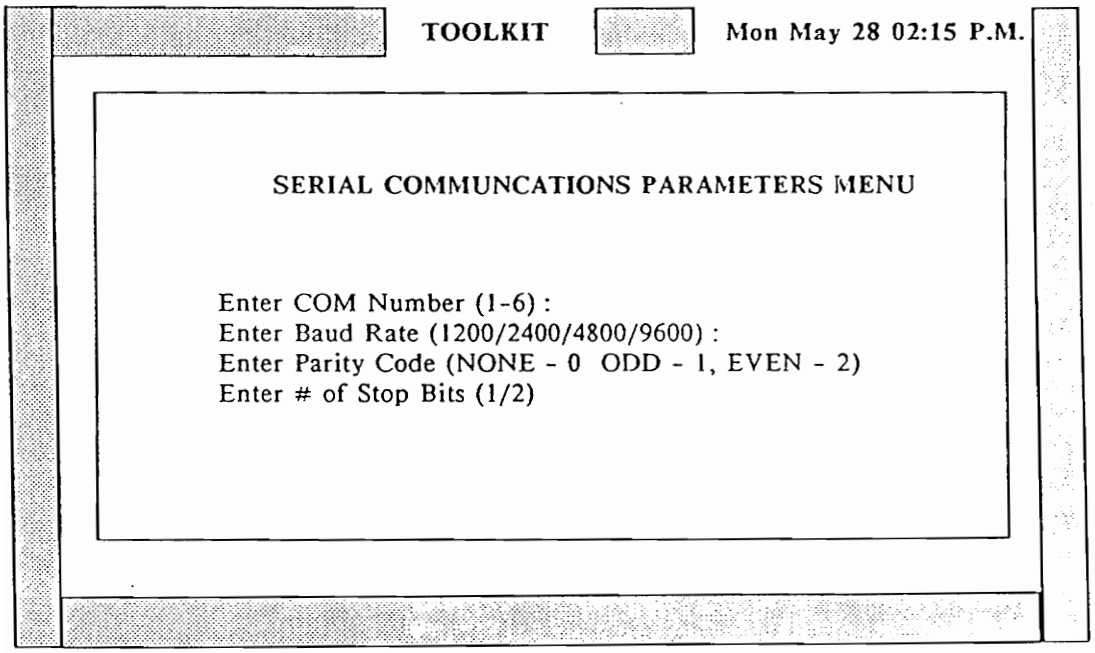


Figure 6.4 Serial communications parameters menu

Robot primitives

The primitives developed for the robot library can be classified into four types. Each type starts the communication transaction with an identifier. The four types and their identifiers are: "execution" (X), "read" (R), "teach" (T) and "program transmit" (N). The protocol sequence for these four function types are shown in Figure 6.5 through Figure 6.9. All the basic functions developed use the routines sendcom(), and receivecom() described earlier (Figure 6.10). Note that before any communication session with the robot controller, the function initrobot() must be called. This routine initializes the serial port with the proper transmission parameters, and then sends two consecutive 'ACK's (hex 06) to the robot controller. The robot controller responds with two consecutive 'EOT's (hex 04) to indicate proper initialization.

a) Execution functions

The execution functions provide host control of the robot control panel. There are two types of execution functions - ones which finish execution immediately, and those which take time to finish execution. The functions execute() and time_delayfn() have been written for the above two cases respectively. The robot execution utilities which

```
DTE:   X        11<0D><0A>
ROBOT:   <13><11          <13>..<13><11>
```

Figure 6.5 Robot "execute" function

```
DTE:   X        31<0D><0A>
ROBOT:   <13><11>          <06>
```

Figure 6.6 Robot "time delayed execute" function

```
DTE:   T        026581E6<0D><0A>
ROBOT:   <13><11>                <06>
```

Figure 6.7 Robot "teach" function

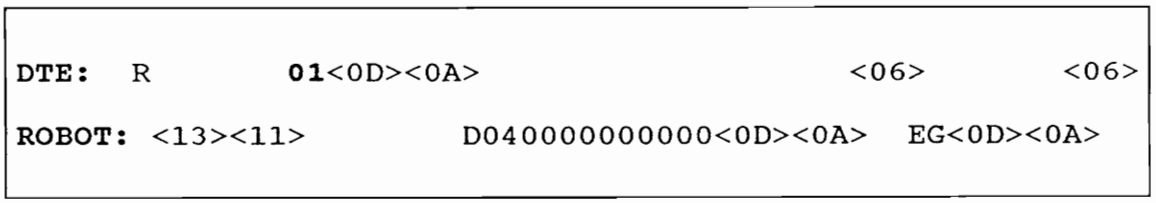


Figure 6.8 Robot "read" function

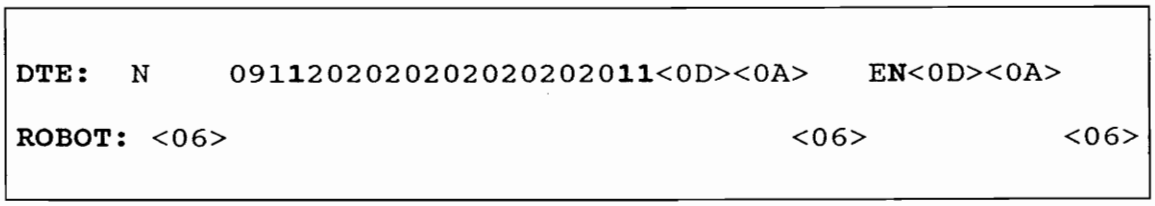


Figure 6.9 Robot "program transmit" function

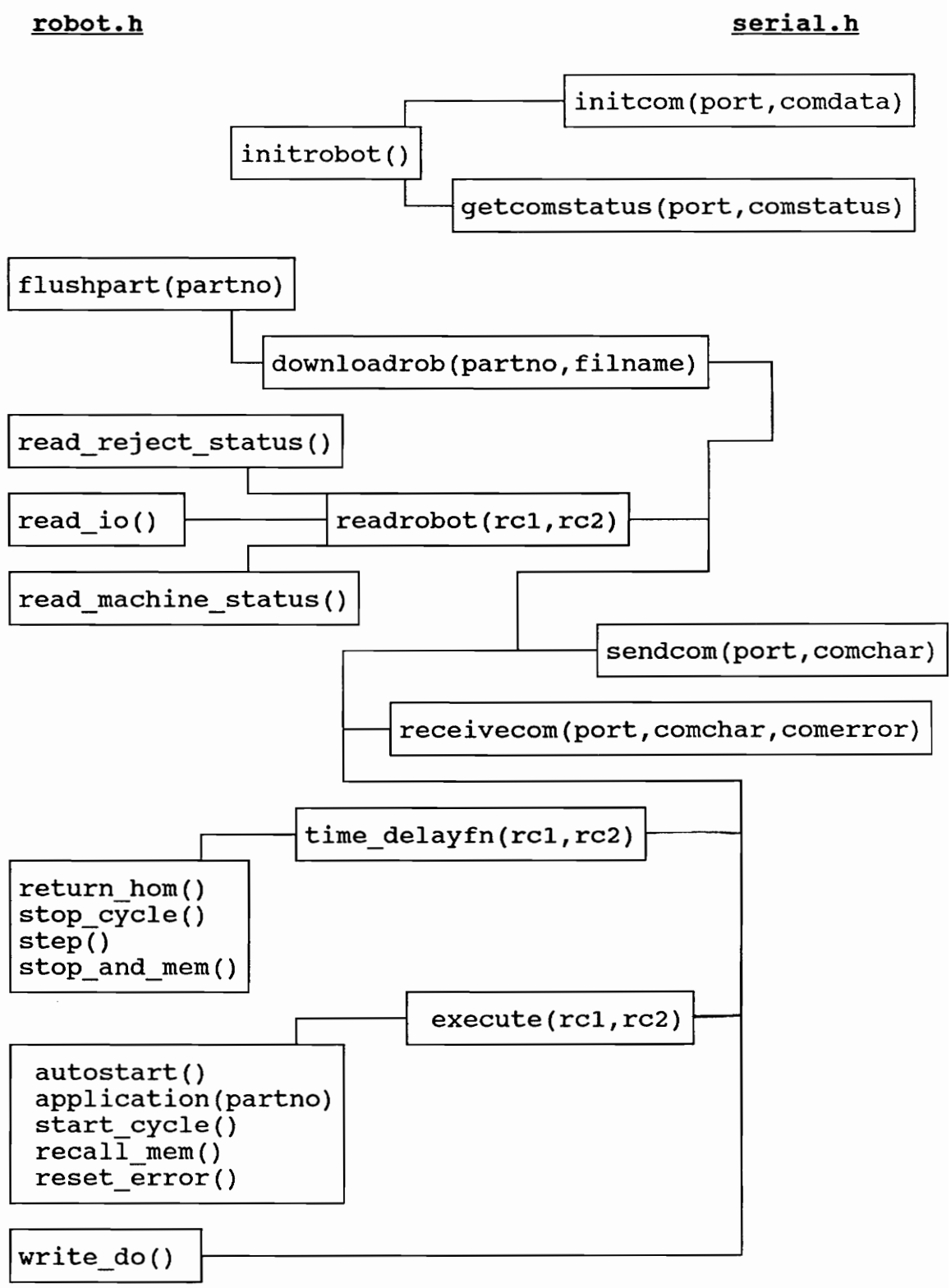


Figure 6.10 Robot functions relationship diagram

call either of the above two functions, pass the execution task code digits into parameters 'rc1' and 'rc2'. The task codes are converted into their ASCII hex equivalents and passed into the routine as character variables 'rc1' and 'rc2'.

int execute(char rc1, char rc2)

The host computer starts the communication by sending an 'X' identifier (hex 58) to the robot controller (Figure 6.5). It then waits for the robot controller to respond with an 'XOFF(hex 13) - XON(hex 11)' sequence. The host then sends the hex task code ('rc1' and 'rc2') for the desired operation followed by a carriage return (Cr - hex 0D) and a line feed (Lf - hex 0A). The host then waits till the controller responds with an 'ACK' (hex 06), which indicates command completion in the case of immediately executable commands.

In the case of the routine time_delayfn(), the operation is similar, with the only difference being that the controller keeps sending an 'XOFF' every 25 seconds till the command is completed, after which it sends an 'XON' (Figure 6.6).

The different execution primitives and their respective task codes are:

1) return_home()	- 11	2) auto()	- 20
3) application(&1)	- 31	4) application(&2)	- 32
5) application(&3)	- 33	6) application(&4)	- 34
7) application(&5)	- 35	8) start_cycle()	- 22
9) stop_cycle()	- 23	10) step()	- 25
11) reset_error()	- 13	12) recall_memory()	- 12
13) stop_and_mem()	- 24		

The functioning of these thirteen functions is briefly described below:

Return_home() sends the robot to the "home" position. Auto() puts the robot controller into the "auto" mode. The robot controller must be in this mode for the rest of the functions to be executable. Application(int *partno) is a routine which enables selection of a robot partition. The parameter passed into the function, 'partno' is a pointer to an integer representing the partition number. In order to start execution of a robot program, it is necessary to download a compiled robot program (file with a '.asc' extension) to a partition of the robot controller. The relevant partition must then be selected for the start_cycle() and stop_cycle() routines, which start and stop the robot cycle, to be executable. Note that

stop_cycle() will only stop the robot program executing at the end of the program. Step() allows the operator to execute the robot program one line at a time, line by line, through keyboard input. Reset_error() resets an error condition of the robot controller. The different types of error conditions possible are: servo error, power failure, overrun error, overtime error, transmission error and data error. In the case of resetting a servo error, it is necessary to manually reset a limit switch OT3 within the robot controller cabinet, before the reset_error() function can be called. When the robot has gone out of its workspace (overrun error), it is necessary to first manually bring the robot arms back into the workspace before reset_error() can be called to deactivate the overrun (OR) LED, and then manually restore manipulator power. Stop_and_mem() and recall_mem() are two complementary routines, useful in stopping/continuing robot program execution at/from a BREAKPOINT in the robot program. A BREAKPOINT statement is used in a robot (.aml) program to provide a pause during program execution, if desired.

The routines return_home(), stop_cycle(), step() and stop_and_mem() call time_delayfn(), since they are not instantaneously executable, while the other execution routines (X identifiers) call execute().

b) Read functions

The read functions of the robot library provide a useful debugging tool during robot error conditions. The three utilities developed under this category are read_io(), read_machine_status(), and read_reject_status(). Routines developed by Guleri [9], to perform these functions were modified. All three routines now call a function readrobot (char rc1, char rc2, int nrec), with different calling variables. The two task code digits are set as characters 'rc1' and 'rc2', while the integer 'nrec' is a variable indicating the number of data records that the calling routine expects to receive from the robot controller. ('nrec' is 2 for read_reject_status() read_machine_status(), 4 for read_io()). The different read primitives and their respective task codes are:

1) read_machine_status() - 01 2) read_io() - 10

3) read_reject_status() - 02

int readrobot(char rc1, char rc2, int nrec)

The function communication sequence is detailed in Figure 6.8. The host computer (DTE) starts the transaction by sending an 'R' identifier (hex 52) to the robot controller. The host then waits for the robot controller to respond with an 'XOFF(hex 13) - XON(hex 11)' sequence, and then sends the hex task code for the desired application

followed by a 'CrLf'. The controller responds with a 'D'(hex 44) or data record (a record is a line of data). The host sends an 'ACK', to tell the controller that the record was correctly received. In the case of read_io(), the robot controller sends two more data records in a similar fashion. The controller then sends an 'E' record to show end of data. The record sent is 'EGCrLf'(hex 45, hex 47, hex 0D, hex 0A). The host sends a final 'ACK' to indicate that the record was received correctly and the requested operation was completed.

The content and interpretation of the data records passed back to the host are now explained for the three functions discussed above.

int read_io()

This function calls readrobot(0x31,0x30,4). It sets the array variable io[9][4] with the status of the digital I/O points read. The elements io[1][1] through io[4][1] indicate the status of the 16 digital inputs, in sets of four, and in reverse numerical order. Each element is a hexadecimal representation of 4 digital points (example: io[1][1] = 'A', in binary means '1010' indicating that digital inputs 16 and 14 are on and that digital inputs 15 and 13 are off). Similarly the array elements io[5][1]

through io[8][1] indicate the status of the digital outputs. If there is an additional digital I/O expansion board, (as in the case of the IBM 7547 in the FMS), then the status of the digital input points is returned in io[1][2] through io[4][2]. The status of digital output points is returned in io[1][3] through io[4][3]. If there is no expansion board, these values are set to 0.

int read_machine_status()

The function calls readrobot(0x30, 0x31, 2), to obtain information on the machine status of the robot. The function sets the global integer variables 'mcst1' and 'mcst2'. These variables represent status codes defined in the AML/E manual [10]. The status codes returned in 'mcst1' are:

1) 80 - Servo error 2) 40 - Power failure

3) 20 - Overrun 4) 10 - Overtime

5) 08 - Transmission error 6) 04 - Data error

The status codes returned in 'mcst2' are:

1) 00 - No data error present 2) 01 - Bus error

3) 02 - Memory test error 4) 11 - Arithmetic error

5) 12 - Programming error 6) 13 - Invalid Op code

7) 14 - Invalid data 8) 15 - Invalid port number

9) 17 - Stack error 10) 18 - Address error

11) 40 - Point out of work space

int read_reject_status()

This routine calls the readrobot(0x30, 0x32, 2) subroutine to obtain a record indicating the reject status of the robot. The function sets the global variable 'rjst' with a reject status code. The reject status codes are:

1) 00 - No reject status
2) 10 - Record format error
3) 15 - Invalid port number
4) 20 - Undefined record
5) 30 - Improper Application startup sequence
6) 40 - Point out of work space
7) 50 - Insufficient memory
8) 51 - Invalid robot type
9) 53 - Invalid Application number
10) 60 - Invalid identifier sent before N record
11) 70 - Xoff time out
12) 80 - Manipulator power off

c) Teach Functions

The only routine developed in this category is write_do().

int write_do(char donum, char onoff)

The routine write_do() teaches the controller new values of digital outputs. This routine provides a useful way to debug and isolate hardware problems. For example, the capability to open/close rams, feeders, as well as the robot gripper can be verified using this software tool. The routine write_do() works in a similar fashion to the other robot communication routines. The specific character flow

sequence is shown in Figure 6.7. The variable 'donum' is the character representation of the hexedecimal digital output point number (example 'donum' = 'A' indicates digital point 10, donum = '0' indicates digital point 16). The character variable onoff indicates whether the digital output point is to be turned on ('1') or off ('0').

d) Program Transmit Functions

Two routines downloadrob(), unload() have been developed. Figure 6.9 gives a specification of the program transmit sequence.

int downloadrob(int *partno, char *name)

Downloadrob() downloads a file 'name' to the application number 'partno', of the robot. This routine is a replica of the routine developed by Guleri [9], with the global character variable 'robotno' modified to be '5' and '7' for the IBM 7545 and IBM 7547 respectively.

int unload(int *partno)

The routine unload(partno) calls the function downloadrob() to send the file "unload.asc" in the aml directory of the linked drive to unload the selected partition 'partno' of the robot controller.

The robot functions discussed above can be exercised by the "toolkit" program using the menu shown in Figure 6.11.

CNC primitives

The routines developed as part of the CNC library are download() and upload(). The functional relationship of the routines is shown in Figure 6.12.

int download(int port, char *filename)

The routine facilitates downloading programs from the host to the DYNA CNC machines. The routine calls the toolkit serial communication routine sendfile(port, comdata, filename), and sets the variable 'comdata' to the global variable indicating the CNC transmission parameters, 'cncdata'. The variables 'port' and 'filename' are passed on unchanged to the sendfile() call.

int upload(int port, char *filename)

The routine facilitates file uploading from the machine to the host controller. This routine calls the toolkit function read_serial(port, comdata). The variable 'port' in the call is unchanged to the variable passed into the upload() routine. The variable 'comdata' is set to the

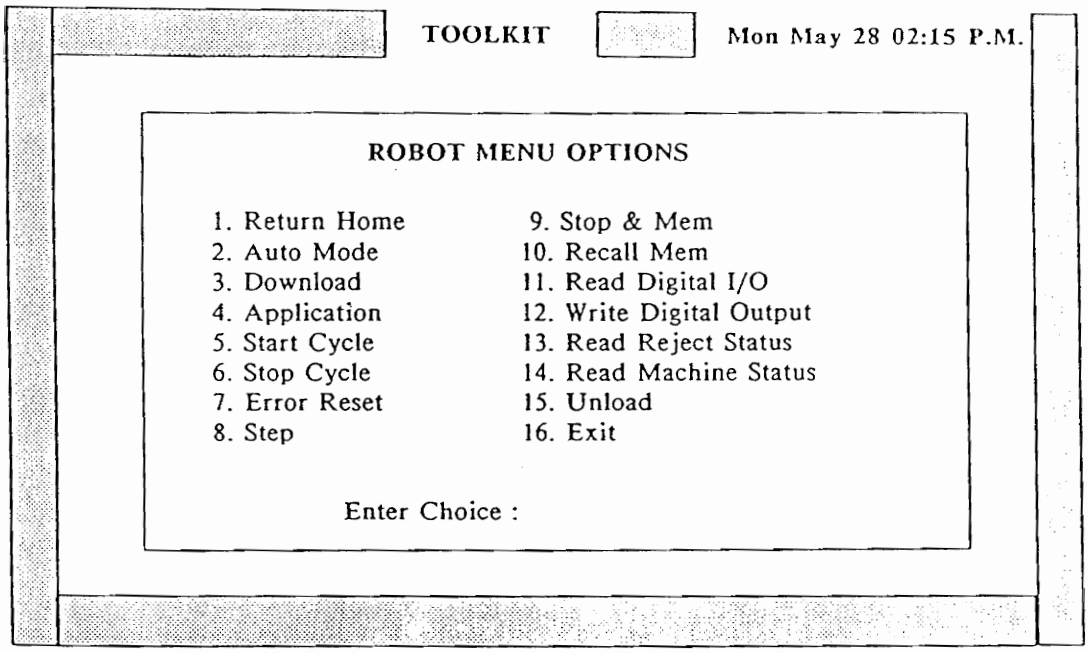


Figure 6.11 Robot menu

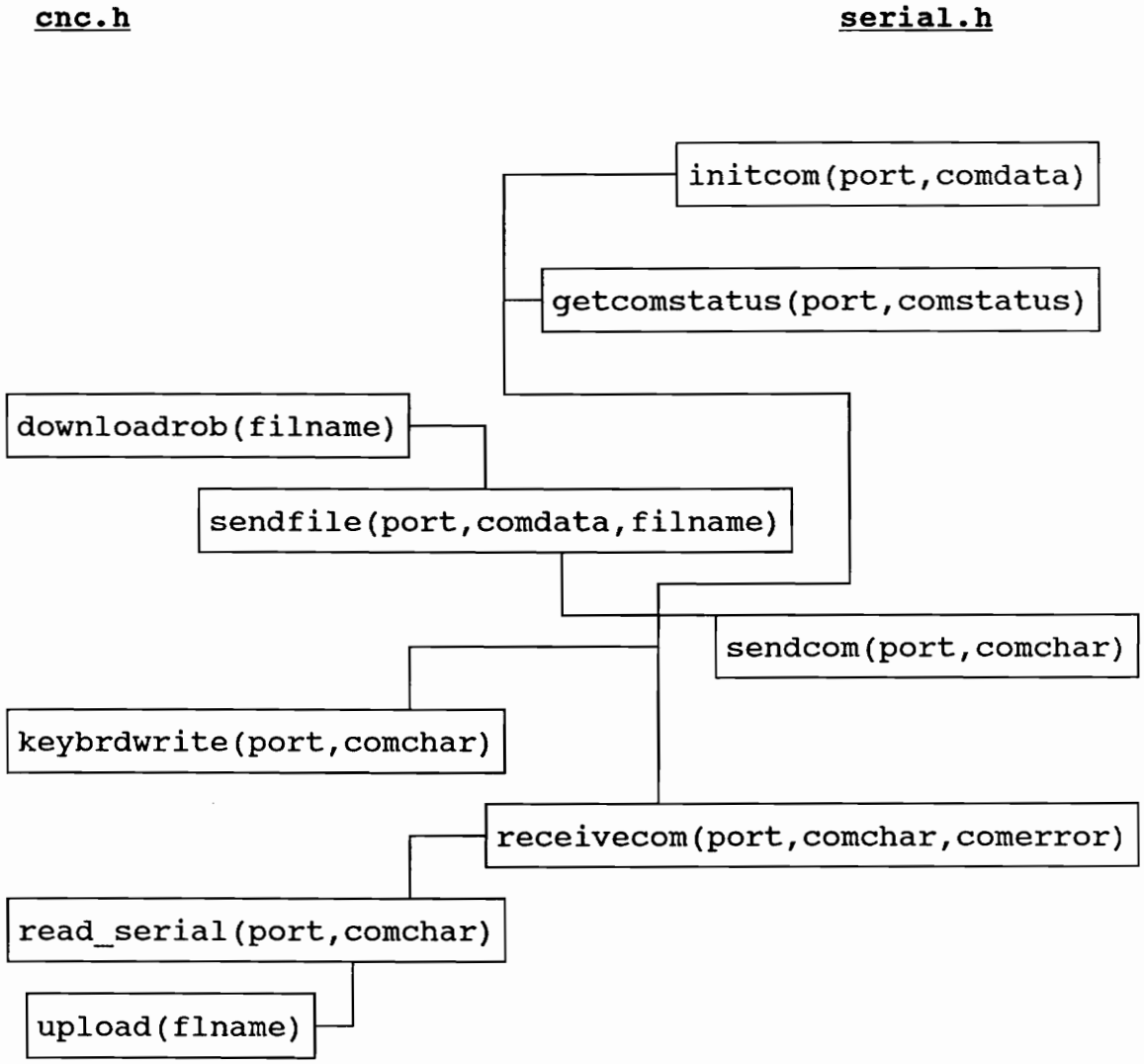


Figure 6.12 CNC functins relationship diagram

global variable 'cncdata'. The contents of the global array combuffer[900] which are reset by the read_serial() function call, is then used to create or overwrite a file 'filename'.

The CNC functions download() and upload() can be exercised by the **toolkit** program as shown in Figure 6.13. Additionally, the menu offers usage of the serial toolkit function keybrdwrite() to download programs created at the keyboard directly to a CNC machine. It must be noted however, that the CNC machine must have been put in the correct mode to receive programs, and that a whole program must be input at the keyboard, before it is downloaded to the CNC machine. This is because the DYNA CNC machines wait to receive an 'END' or a '900' statement, before accepting the program.

The toolkit also offers controlling the start of the execution of the DYNA CNC machines. A detailed description of how this handshake is performed, using the PLC, and workcell digital I/O, is given in Chapter 7.

Digital I/O routines for DATA TRANSLATION 2808 board

The two routines dtread() and dtwrite() have been developed in order to enable the workcell to read digital

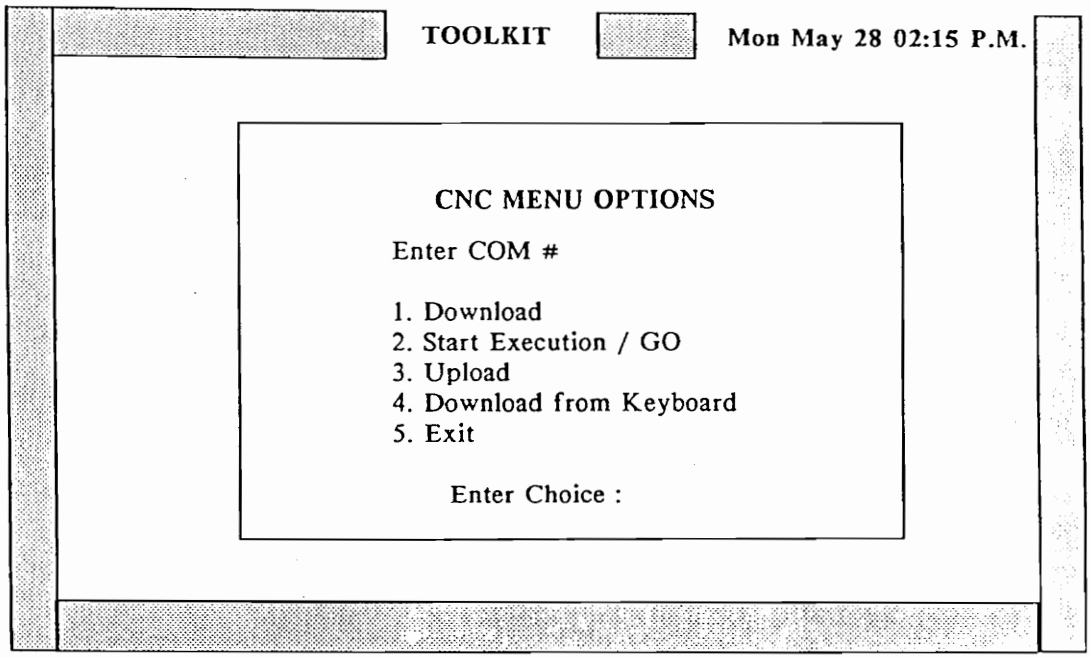


Figure 6.13 CNC menu

inputs or write digital outputs respectively. The DATA TRANSLATION 2808 board has two ports (port 0 and port 1). Each port has 8 bits. The range of values for each port is 0 (all 8 bits off) to 255 (all 8 bits on). The routines follow the procedure suggested in the DATA TRANSLATION 2808 manual [3]. The base address used is the one suggested - 2EC hex. The routines can be exercised by the "toolkit" program, as shown in Figure 6.14.

int dtwrite(int value)

The routine calls subroutines m11() and m13(int value), which are translations of the programs 'MANEP11.BAS' and 'MANEP13.BAS', in the DATA TRANSLATION manual. The allowable range of the variable 'value' in m13() and dtwrite() is 0 to 255.

When this routine is exercised by the "toolkit" program, the user is asked for input on which bit he wants turned on/off. The variable 'value' to be outputted to the I/O port, by using the user input and the previous status of the 8 bits stored in the global variable MASKDT. It is necessary to refresh the global variable MASKDT after every write operation, so that one can control any particular bit without affecting the status of the other 7 bits.

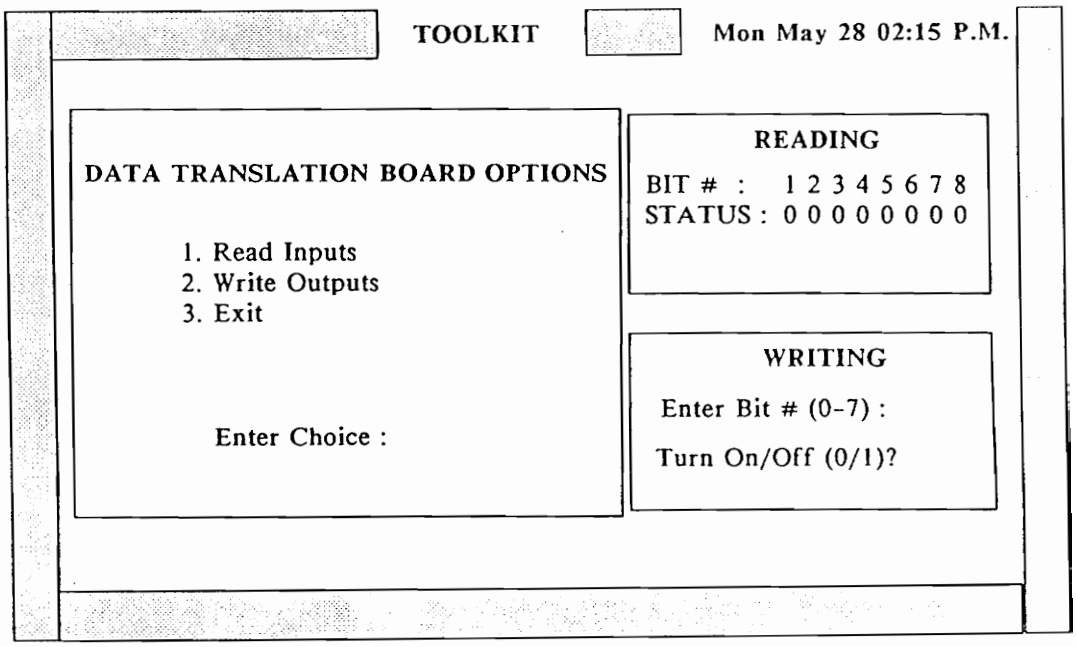


Figure 6.14 DATA TRANSLATION board menu

int dtread()

The routine returns an integer variable 'value', indicating the status of 8 bits controlled by the DATA TRANSLATION board.

The "toolkit" program uses the returned 'value' to calculate the status of the individual bits.

Digital I/O routines for DIO64 board

The DIO64 board has 8 ports. Each port has 8 bits. The range of values for each port is 0 (all 8 bits on) to 255 (all 8 bits off). In the current setup only ports 0 to 3 have been connected. The base address used for the DIO64 board is 208 hex. The "toolkit" program can be used to exercise the two routines described above, as shown in Figure 6.15.

int d64write(int portno, int value)

The routine d64write(portno, value) has been written following the guidelines in DIO64 manual [11]. The function performs an operation similar to that performed by the routine dtwrite() developed for the DATA TRANSLATION board. The variable 'portno' indicates the port number, and has an allowable range of 0 to 7.

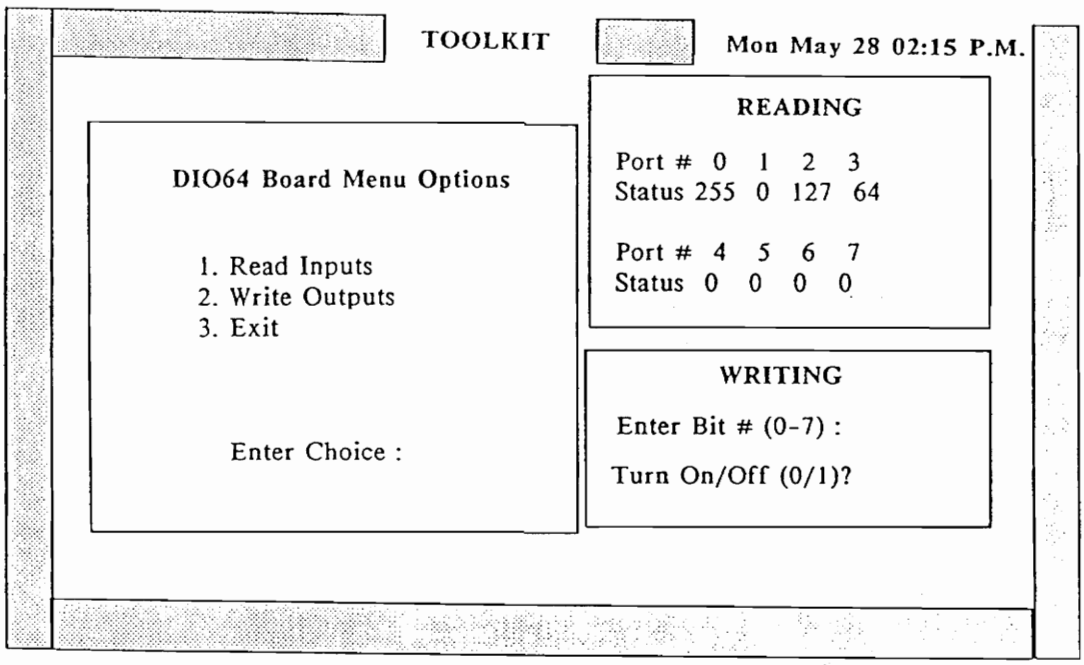


Figure 6.15 DIO64 board menu

int d64read()

The routine d64read() reads the status of all 8 ports of the DIO64 board. It sets the global variable array 'd64value[8]' with the values returned by each port (example: d64value[0] = 127 indicates that port 0 has all bits except bit 7 turned on).

7. MACHINING WORKCELL

The machining workcell is responsible for all machining operations associated with the FMS products. Hence kits, consisting of various wax blocks arrive at the cell for processing. The robot transports these blocks to and from the CNC milling machines, which perform the desired machining operations. Coordination and control of the cell activities is the responsibility of the cell controller.

Hardware

The machining workcell controller is an AT&T 6300 computer. The controller co-ordinates the activities of equipment in its domain, namely

1) an IBM 7545 robot,
2) two DYNA CNC milling machines,
3) fixtures on the CNC machines, and
4) a TI 565 PLC.

The machining workcell controller has a STARLAN board, a FASTCOM4 multiplexer board, and a DATA TRANSLATION 2808 I/O board. The serial port (COM1) on the mother board, is connected to the robot controller by means of a RS-232C cable.

The FASTCOM4 multiplexer board enables usage of 4 additional serial ports (COM 3-6). RS-232 cables link COM3 and COM4 to the DYNA machines. COM5 and COM6 are unused. In order to use COM 3-6, it is necessary to load the device driver "commbios" via the "autoexec.bat" file. Additional information on this aspect is given in Appendix A. The RS-232C cable configurations for the robot and the DYNA machines are given in Appendix B.

The DATA TRANSLATION 2808 I/O board facilitates both analog and digital input/output. In the FMS setup, analog I/O is not being used. The digital I/O is employed to connect the workcell controller to the robot and the TI programmable controller. The DATA TRANSLATION board is interfaced to an Optomation PB16T 16-position I/O module mounting rack (Figure 7.1). The I/O modules are used to convert from the TTL (5 V DC) logic of the computer to higher industrial voltage levels (24 V DC), while keeping the host computer isolated. The DATA TRANSLATION board has

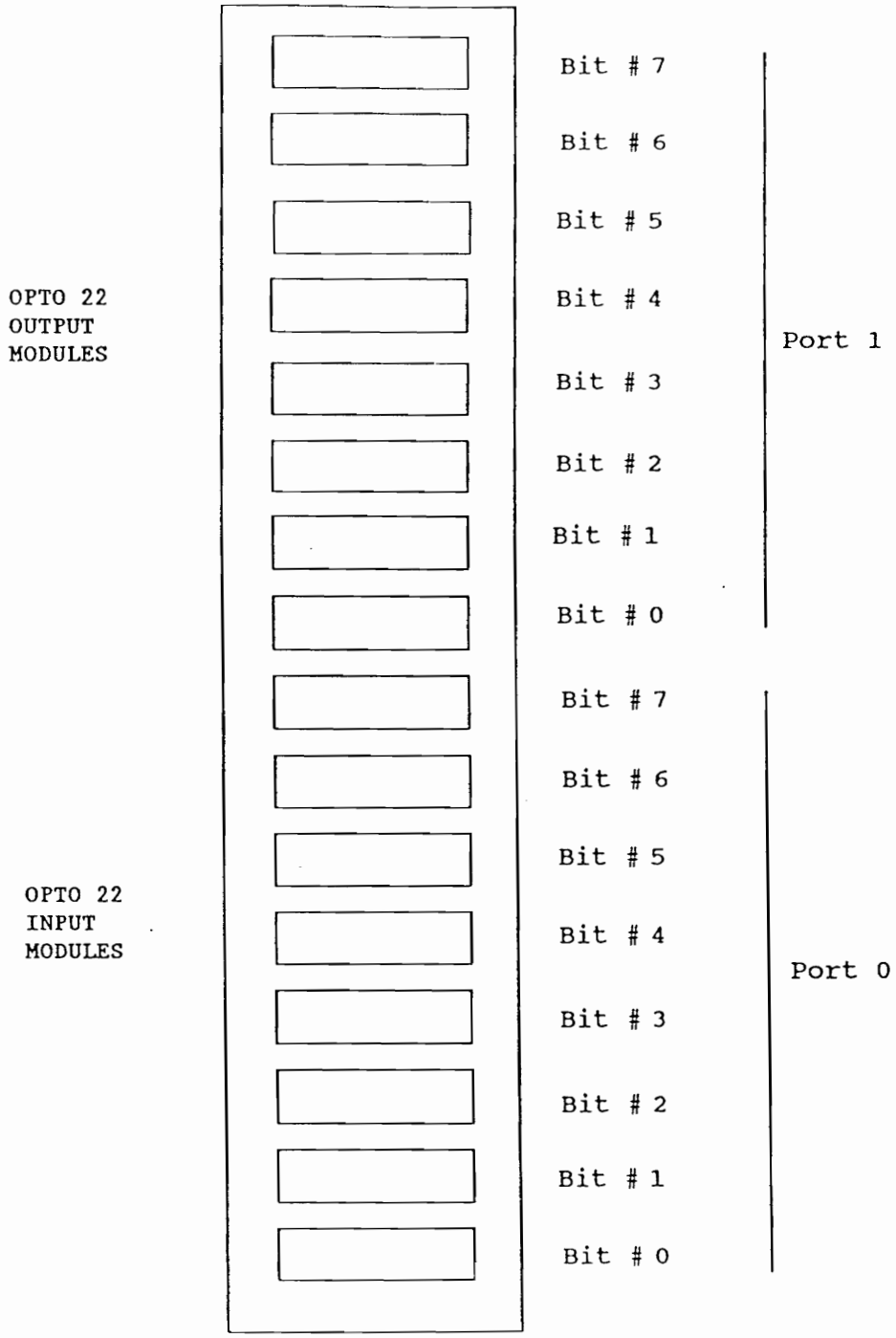
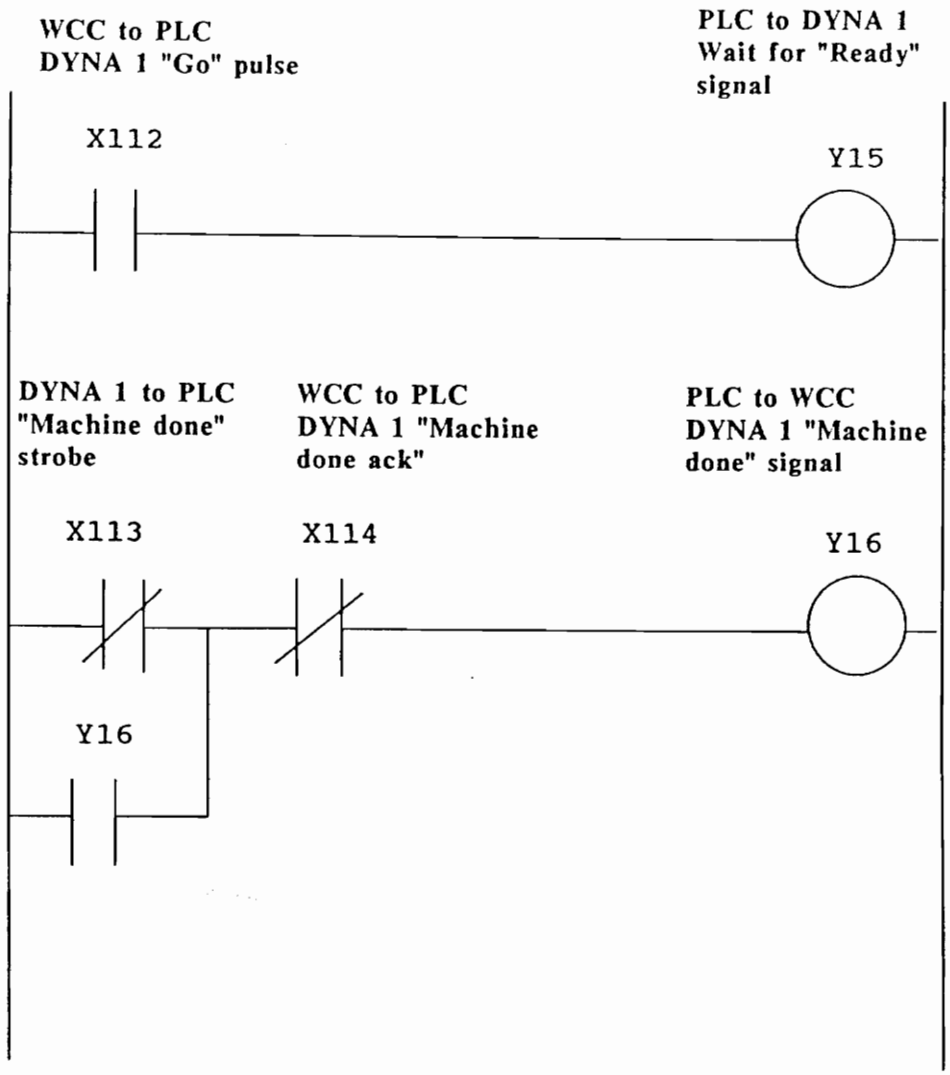


Figure 7.1 OPTO22 mounting rack of machining workcell

2 programmable digital I/O ports of 8 bits each. The lower 8 positions on the mounting rack have 8 OPTO22 DC input modules and are connected to Port 0, while the upper 8 positions on the mounting rack have 8 OPTO22 DC output modules and are connected to Port 1.

The system has been designed so that the inputs to the cell controller are from robot status digital outputs and PLC handshaking outputs. The outputs from the controller are handshakes to the PLC. The DYNA machines are connected to the TI PLC by two discrete I/O lines - one input and one output (Appendix B). These input/output lines are used to synchronize the operations of the milling machines with other workcell devices. At the beginning of a DYNA program, a CONTROL 3 statement is inserted that will cause the machine to stop execution and wait for a pulse from the workcell controller. This pulse is sent from the controller to the PLC and then to the DYNA milling machine. A CONTROL 2 statement is inserted at the end of the program, to cause the DYNA to send a pulse to the PLC indicating that the program is done. Since this pulse has a duration of only 100 milliseconds, the PLC latches the pulse so that it can be read by the workcell controller when it is ready to check the state of the machines (Figure 7.2). Otherwise the pulse might be missed if the controller did not look for it



X113 is normally high

Figure 7.2 Handshaking ladder

during its 100 millisecond duration. A digital output from the workcell control is used to clear the state latch in the PLC when the signal has been recognized. Since the DYNA sends a 5 V DC pulse, while the PLC expects a 24 V DC input, it is necessary to have a step-up relay on the CNC discrete output line.

The air cylinders that are used in the vices on the CNC milling machines are activated by robot outputs, while the limit switches on these machines activate robot inputs. The robot however does not communicate directly with the CNC machines.

Software

The machining workcell controller software menu system **macwcc** provides the user the option of interacting directly with workcell devices from the workcell controller or passing control to the systems controller. The sequence of menus are given in Figure 7.3 through Figure 7.5. Regardless of whether the workcell is to be controlled

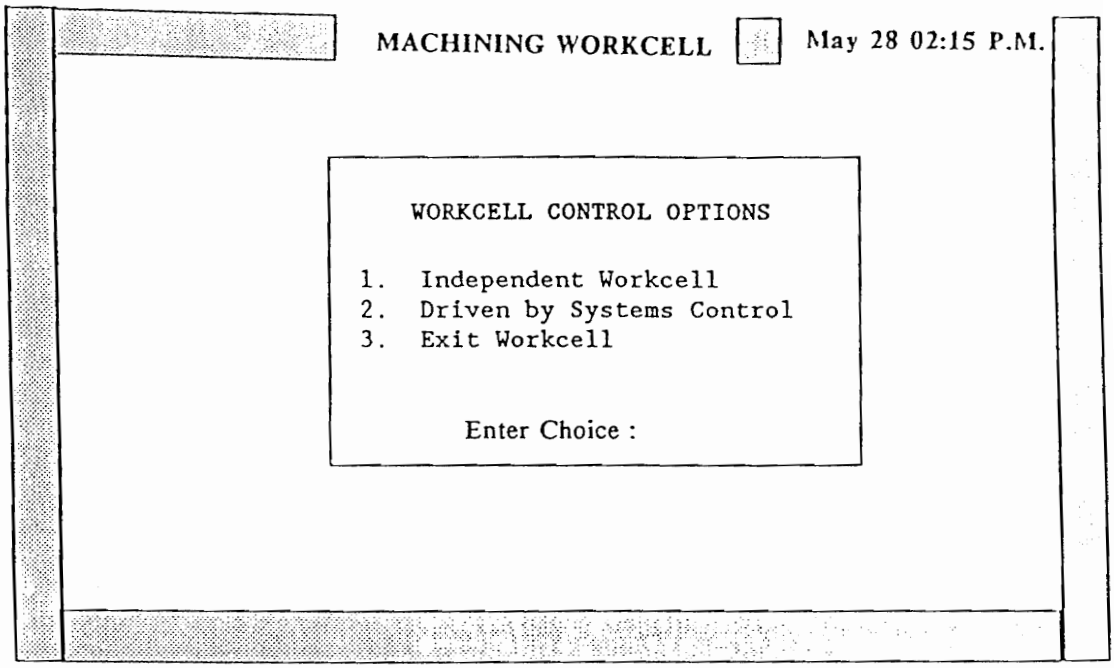


Figure 7.3 Machining workcell control options

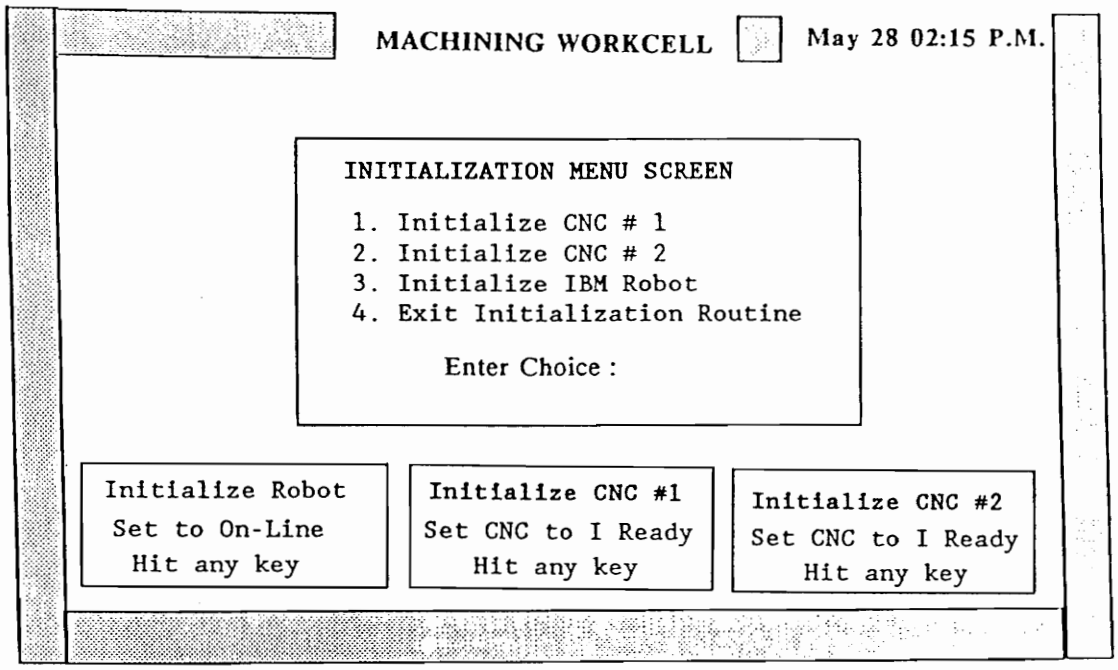


Figure 7.4 Machining workcell initialization menu

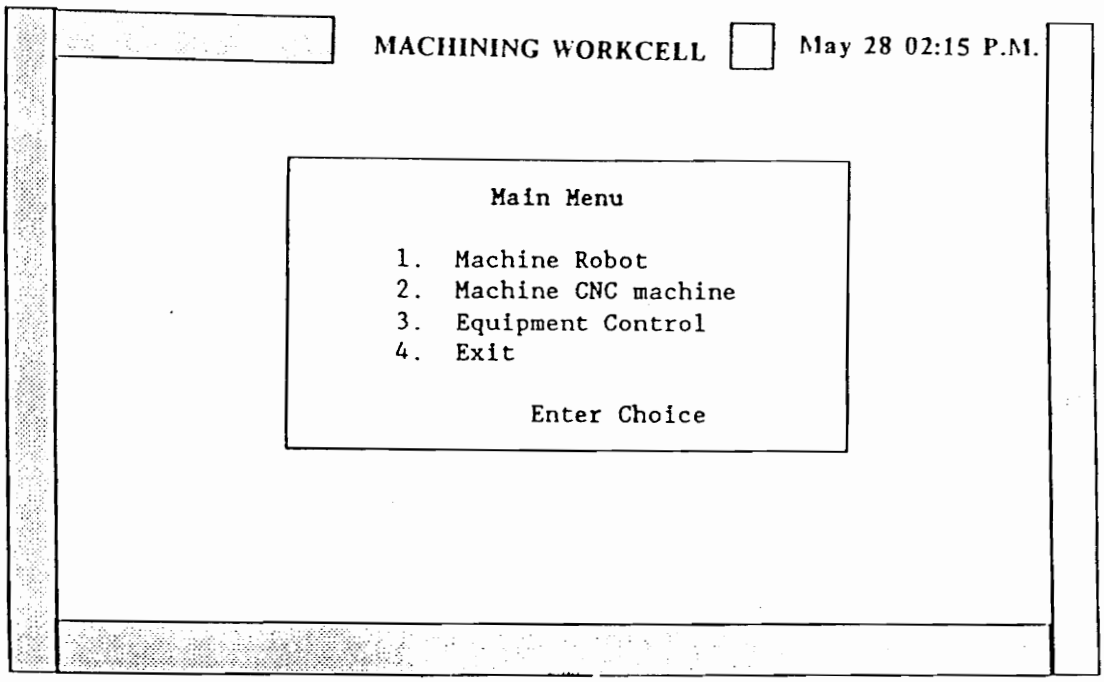


Figure 7.5 Machining workcell main menu

locally or via systems control, the user is first asked to initialize the different pieces of equipment before proceeding (Figure 7.4). In the case of the robot, this involves sending it to the "home" position and placing the robot controller in the "auto" mode. In the case of the CNC machines, this involves manually setting up the machines to receive programs and then establishing the origin of the reference coordinate system. The respective routines used to achieve the above tasks are return_home(), autostart() and download(ncnumber, filename).

If the user chooses to run the workcell from the local controller, three options are presented: machine a robot, machine a CNC or perform primitive control of equipment within the workcell. This is illustrated in Figure 7.5. The routines developed for the above are mirobot(), micnc(), and equipment(). The mirobot() and micnc() routines are "macro" routines, which cause a sequence of tasks to be initiated through communication sequences between the workcell control and various workcell devices. The routine equipment() provides a menu to access the primitives of the robot or either CNC machine.

The typical sequence of operations and handshakes involved in the machining workcell can be described by

considering the initial steps taken in the machining of a wax CNC machine (Table 7.1). The cell controller downloads a program, to load raw material onto the CNC machine, to the robot controller. The execution of the program is then started. After the CNC machines have been loaded, and the robot has moved away to a safe position, the IBM 7545 robot sends a 'done' signal to the cell controller. On receiving this signal, the controller writes a digital output which is picked up as an input by the PLC. The ladder logic in the PLC ensures that the PLC then sends a synchronous pulse to the DYNA CNC machine. The DYNA which all along had been waiting for this 'GO' signal, (at a CONTROL 3 statement), then starts execution of the machining section of DYNA code. After finishing machining, the DYNA sends back a synchronous pulse to the PLC with a CONTROL 2 statement. The PLC then indicates to the cell controller that the DYNA has finished machining. The cell controller now "tells" the robot to go and pick up the part from the CNC machine.

Handshakes between the different pieces of equipment are necessary to ensure proper sequencing of parallel operations by the different devices, after these operations have completed. For example, it is necessary that the DYNA machine start machining the part only after the robot has

TABLE 7.1 Typical Handshake sequence

downloadrob(partno, filename)

download(ncnumber, filename)

start_cycle(partno)

stop_cycle()

'Robot done' signal : Robot to Cell Controller.

'DYNA #x go signal' : Cell Controller to PLC.

'DYNA #x GO pulse' : PLC to DYNA #x.

'DYNA #x DONE pulse' : DYNA #x to PLC.

'DYNA #x machine done signal' : PLC to Cell Controller.

'DYNA #x machine done ack' : Cell Controller to PLC.

downloadrob(partno, filename)

start_cycle(partno)

stop_cycle()

'Robot done' signal : Robot to Cell Controller

moved out of the way, after loading the part on the CNC. Likewise, the robot should come to pick up the part only after the CNC has finished machining. It should be noted that the present system has been designed to ensure a hierarchical handshake protocol. The equipment in the cell do not communicate directly with each other, but rather via the cell controller itself. Thus the robot and the DYNAs communicate with each other via the cell controller and the PLC.

It must be noted that the section of PLC ladder for the handshaking in the machining workcell, is independent of the rest of the ladder developed for the material handling workcell. In fact, the two sections of ladder code could well have been implemented on two different PLCs, as shown in Figure 4.4.

Mirobot() and micnc() perform the following sequence of tasks:

1) Download robot programs to the robot controller, which perform the function of transporting parts between the pallet and the CNC machines.
2) Execute the first sequential robot program of loading the CNC machine. Using the handshake policy, control the start of CNC machining when the robot is out of the

way of the CNC machine, and similarly control the start of the robot cycle to come and pick up parts, after machining is completed. Repeat the same sequence of tasks for the next operation in the cycle.

The supporting robot and CNC files are described in Table 7.2. It must be noted that while the links of the robot and CNC machine products are taken only to the finish cut DYNA machine (which has a smaller tool diameter), the bases of both the robot and the CNC machine are taken first for a rough cut to the CNC machine with a larger tool diameter and then taken to the other CNC machine for a finish cut. Table 7.3 illustrates the specific functions, files and signals generated during the execution of mirobot().

When the machining workcell controller (WCC) has passed control to the systems controller, the functions sysconinit() and machinecmd() are executed. Sysconinit() forces the operator to initialize the equipment (Figure 7.3). The function machinecmd() causes the WCC to poll the linked directory for the file 'mwcin.dat'. When the systems controller wishes the machining cell to perform a task, it writes the macro command name (Table 7.4 enumerates the set of possible commands) into the above file. When the

TABLE 7.2

Robot Supporting files for Machining workcell

The nomenclature used for the robot files is self-explanatory - the first character indicates product type, the next two indicate part type, the fourth and fifth characters indicate starting location and the last two characters indicate destination location. The '.asc' extension indicates that the file is a compiled '.aml' file with the same name. Note that only the compiled versions of the robot program can be downloaded to the robot controller.

Rbspld2.asc - Transport robot base from pallet to DYNA 2

Rl1pld1.asc - Transport robot Link 1 from pallet to DYNA 1

Rl1d1pl.asc - Transport robot Link 1 from DYNA 1 to pallet

Rl2pld1.asc - Transport robot Link 2 from pallet to DYNA 1

Rl2d1pl.asc - Transport robot Link 2 from DYNA 1 to pallet

Rbsd2d1.asc - Transport robot base from DYNA 2 to DYNA 1

Rbsd1pl.asc - Transport robot base from DYNA 1 to pallet

CNC supporting files for Machining workcell

Robbasd2.cnc - Robot base on DYNA 2 (Rough Cut)

Robarm1.cnc - Robot arm1 on DYNA 1

Robarm2.cnc - Robot arm2 on DYNA 2

Robbasd1.cnc - Robot base on DYNA 1 (Finish Cut)

Cncbasd2.cnc - CNC base on DYNA 2 (Rough Cut)

Cncarm.cnc - CNC arm on DYNA 1

Cncbasd1.cnc - CNC base on DYNA 1 (Finish Cut)

TABLE 7.3 Breakdown of macro mirobot() of the Machining workcell

Specifically, **mirobot()** performs the following sequence of tasks: (The tasks performed by the respective robot and CNC files is explained in Table 7.2)

1) Downloads 5 files namely 'Rbspld2.asc', 'Rl1pld1.asc', 'Rl1d1pl.asc', 'Rl2pld1.asc', 'Rl2d1pl.asc', to partitions 1 through 5 of the robot controller.
2) Downloads file 'Robbasd2.cnc' to DYNA 2 and 'Robarm1.cnc' to DYNA 1.
3) Executes Partition 1 of the robot - transports robot base from pallet to DYNA 2 for rough cut.
4) The cell controller receives a 'Robot Done' signal.
5) The cell controller writes a digital output, which is an input to the PLC.
6) The PLC then sends a GO pulse to DYNA 2 which was waiting at a CONTROL 3 statement to start machining.
7) Executes Partition 2 of the robot - transports robot link # 1 from the pallet to DYNA 1.
8) Repeat steps 4), 5) and 6) replacing DYNA 2 with DYNA 1.
9) When DYNA 1 finishes machining, it sends a pulse to the PLC with a CONTROL 2 statement.

TABLE 7.3 continued

10) On receiving the above pulse, the PLC sends a 'DYNA 1 machine done' signal to the cell controller.
11) The cell controller then sends a machine done acknowledgement to the PLC.
12) The cell controller then executes Partition 3 of the robot - transports the machined link 1 from DYNA 1 to the pallet.
13) Download 'Robarm2.cnc' to DYNA 1.
14) Repeat Steps 7) to 12) replacing Partition 2 and 3 with Partitions 4 and 5 respectively for link # 2.
15) Repeat steps 9) to 11) for DYNA 2.
16) Flush Partitions 1 and 2.
17) Download robot files 'Rbsd2d1.asc' and 'Rbsd1pl.asc' to Partitions 1 and 2 of the Robot Controller.
19) Download 'Robbasd1.cnc' to DYNA 1.
20) Repeat steps 7) to 12) for DYNA 1 handshaking with robot Partitions 1 and 2.
21) Send 'Robot done' signal to the cell controller.

TABLE 7.4

System control commands to Machining workcell

Command	Task
mirobot	Machine robot
micnc	Machine CNC
equipment	Control primitives of equipment (robot and CNC machines)

machining WCC finds the file in the linked directory, it reads this file, and executes the routine with the same name as the command it received (example: when the systems controller writes "mirobot" in file 'mwcin.dat', the function mirobot() is executed by the machining WCC). In the case when the command read in the file 'mwcin.dat' is "equipment", then the WCC polls the file 'mwc2in.dat' for specific information required for primitive control of devices in the workcell (example: the strings "downloadrob" "1" " rbspld2.asc" in file 'mwc2in.dat' will cause the machining WCC to download file 'rbspld2.asc' to partition 1 of the robot controller). When the WCC has read a task command, it erases the command input file 'mwcin.dat' or 'mwc2in.dat'). It then creates a status output file 'mwcout.dat' and writes a "busy" into this file. The file 'mwcout.dat' is monitored by the systems controller in a similar manner. When the task has been completed by the device in the workcell, the WCC sends a "done" status to the systems controller. The WCC then continues polling the common linked directory for further command inputs in the file 'mwcin.dat'. In this way the systems controller and workcell controller are able to communicate with each other through appropriately addressed files (envelopes) stored in a common linked directory (mailbox). The systems controller is able to achieve control of the workcell capabilities, as

well as have status feedback from the workcell. Similar to the WCC, the systems controller erases the status file 'mwcout.dat', after it has been read. It then continues searching in the linked directory for new status information.

8. ASSEMBLY WORKCELL

The assembly workcell is responsible for replenishing raw material feeder brackets, kitting pallets and assembling the final products. All operations are performed by the robot using the fixturing in the cell.

Hardware

The equipment under the domain of the assembly cell, are an IBM 7547 robot, raw material feeders, and assembly fixtures. The assembly workcell controller is an AT&T 6300 computer with a STARLAN board and a DATA TRANSLATION 2808 board. Port 0 on the DATA TRANSLATION board is currently linked via a PB16T mounting rack with OPTO22 modules to robot status outputs. The serial port (COM1) on the mother board is connected to the robot controller by means of a RS-232C cable. The IBM 7547 robot controller has within it an input/output additional expansion unit for additional digital I/O. There are thus 32 inputs and 32 outputs for this robot. Only 4 inputs are being currently used however [2].

Software

The assembly workcell controller software menu system **asswcc** functions much like the software system for the machining workcell. The user is presented with a menu (Figure 8.1), which requests input designating the desired mode of operation - locally controlled or controlled by the systems controller. Both modes of operation will require initialization of the robot and this is done through the menu given in Figure 8.2. The routines developed to accomplish the above tasks are: return_home() and autostart().

Local control of the assembly workcell is accomplished through the menus given in Figure 8.3. Eight different tasks can be performed. Seven of these tasks are executed through command/communication macro routines, as described in the machining workcell software description. The last task, 'Primitive Control', provides primitive control access to the robot controller.

Asrobot() and ascnc() routines are used to assemble the robot and CNC machines. To build robot and CNC kits, burobot(), and bucnc() are executed. Lorobot(), locnc() and lolink() cause the robot to replenish raw material feeders

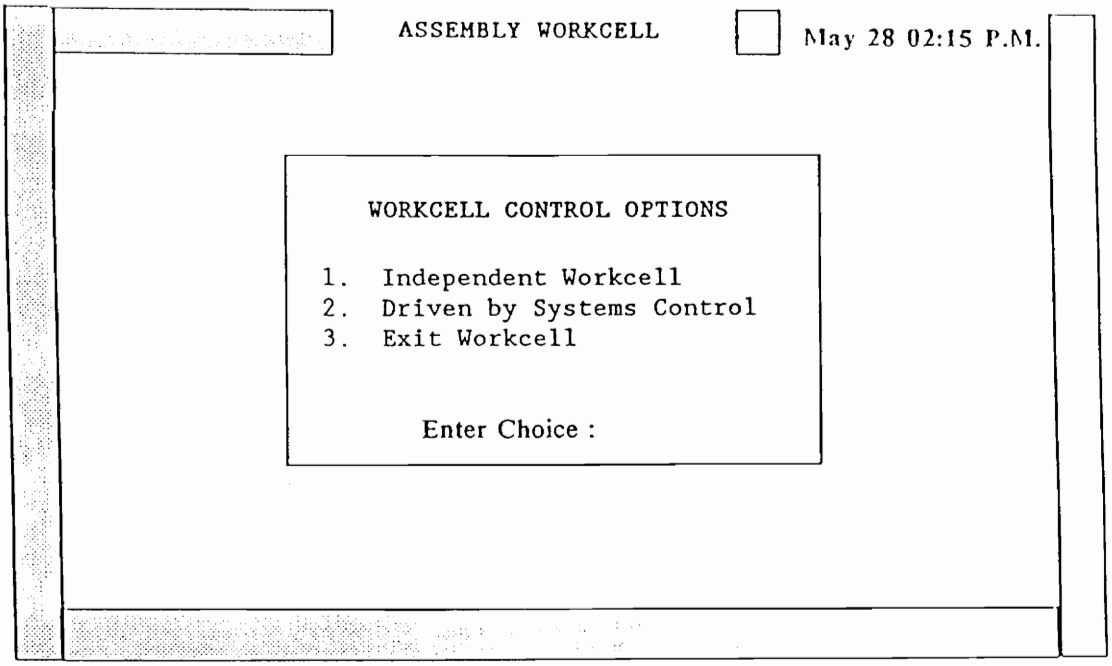


Figure 8.1 Assembly workcell control options

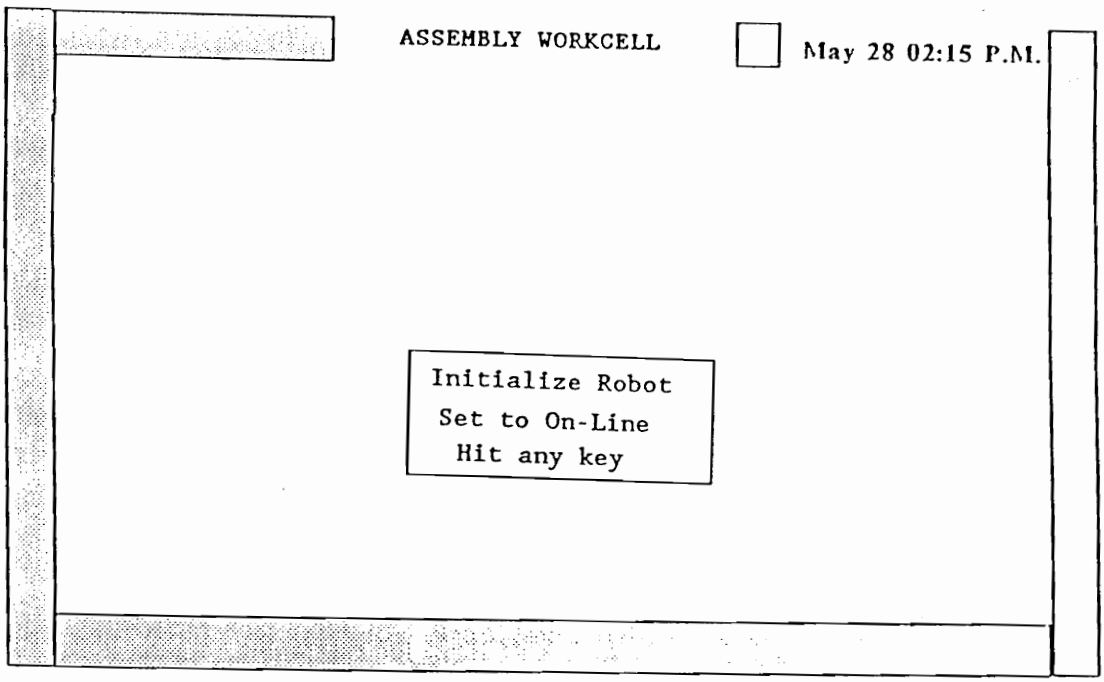


Figure 8.2 Assembly workcell initialization menu

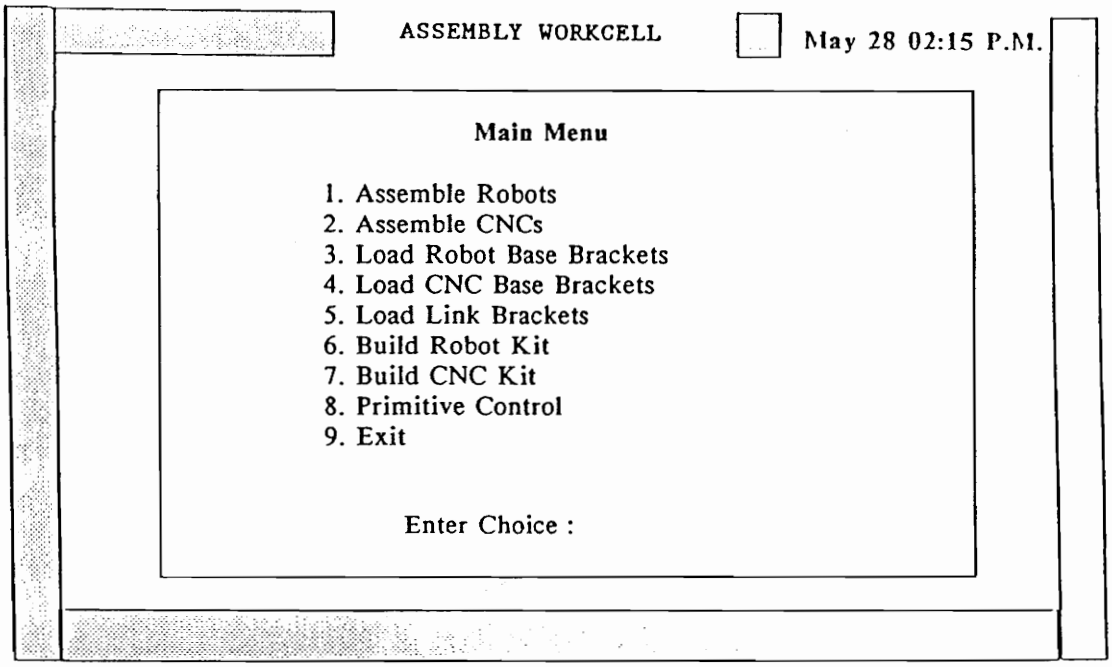


Figure 8.3 Assembly workcell main menu

for robot bases, CNC bases and links respectively. These macros are routines which download and execute robot files with the same names as the routines themselves, with an '.asc' file extension. Table 8.1 lists the various robot files.

When the assembly workcell is driven by the systems controller, the communications involved are similar to that discussed in Chapter 7 for the machining workcell. The workcell controller polls the file 'awcin.dat' for commands from the systems controller, and writes status information into the file 'awcout.dat'. File 'awc2in.dat' is used by the systems controller to control the primitives of the assembly workcell robot. Table 8.2 summarizes the commands from the systems controller to the assembly workcell.

TABLE 8.1

Robot supporting files for Assembly workcell

The nomenclature used for the robot files are indicative of the tasks they perform. Also share same name as system control task commands to this workcell.

Asrobot.asc - Assemble robot

Ascnc.asc - Assemble CNC

Burobot.asc - Build robot kit pallet

Bucnc.asc - Build CNC kit pallet

Lorobot.asc - Load robot base bracket

Locnc.asc - Load CNC base bracket

Lolink.asc - Load link bracket

TABLE 8.2

System control commands to Assembly workcell

Command	Task
asrobot	Assemble robot
ascnc	Assemble CNC
burobot	Build robot kit
bucnc	Build CNC kit
lorobot	Load (Replenish) robot base brackets
locnc	Load (Replenish) CNC base brackets
lolink	Load (Replenish) link brackets
equipment	Control primitives of robot

9. MATERIAL HANDLING

The material handling workcell is responsible for storage, transportation, and inspection. Currently, since the vision system and the AS/RS have not yet been incorporated, its basic task is transportation of pallets between the operator interface, storage, assembly workcell and machining workcell.

Hardware

The conveyor used for transportation is a Shuttleworth conveyor system. Control and status monitoring of conveyor motion is achieved with TI565 PLC inputs and outputs. The material handling workcell controller is an AT&T 6300 computer with a STARLAN board and a DIO64 board manufactured by Industrial Computer Source. The DIO64 board has 8 ports of 8 bits each (totalling 64 I/O points), and configured into 4 sets. Each set has 16 bits of programmable input/output. Currently 2 sets of bits are being used to interface with 2 Optomation PB16A mounting racks (Figure 9.1). Set 1, which has Port 0 (lower 8 positions) and Port 1 (upper 8 positions), is connected via

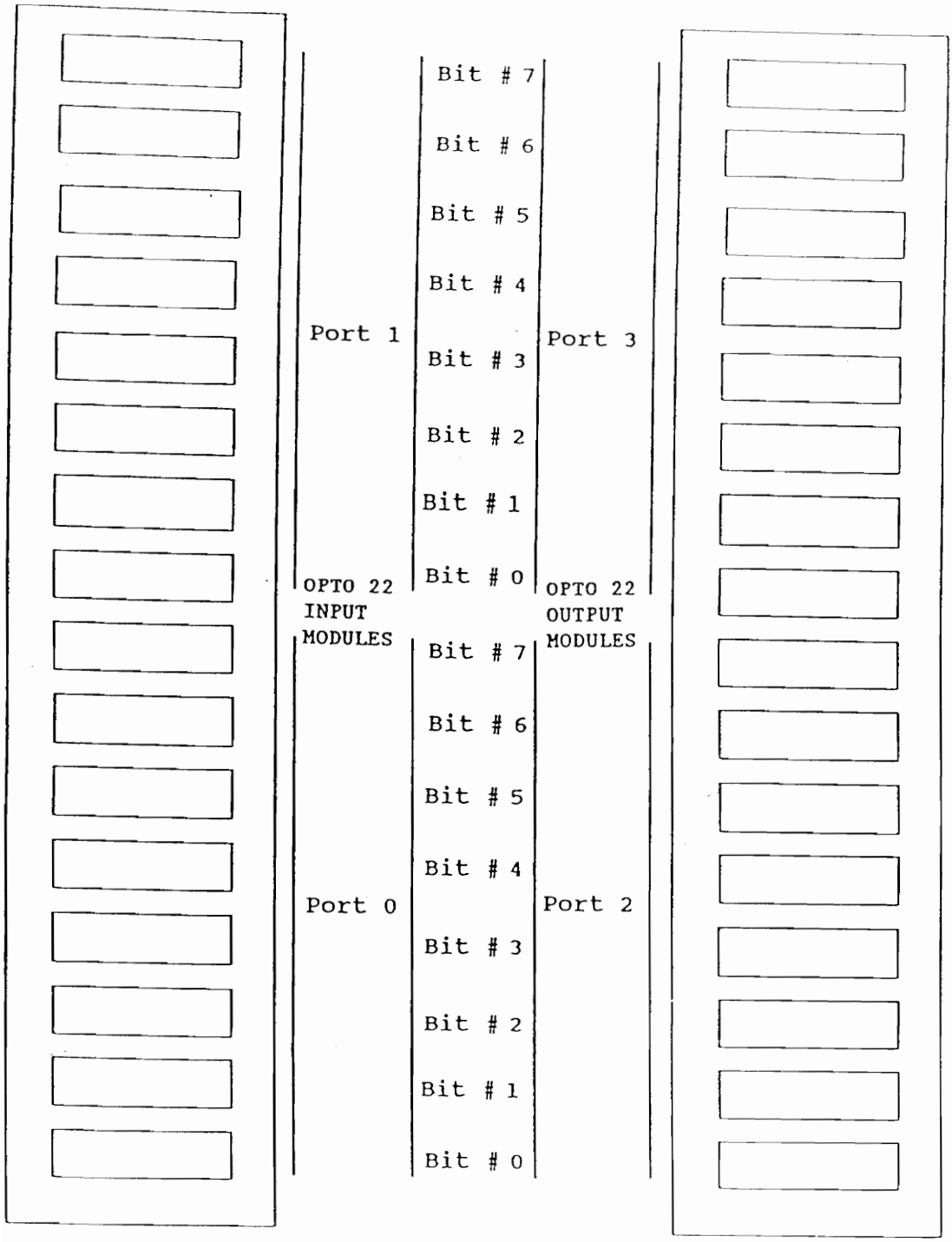


Figure 9.1 OPTO22 mounting racks of material handling

ribbon cable to the mounting rack with digital input modules. These input modules are activated by PLC outputs for conveyor and AS/RS status. Set 2, which has Port 2 (lower 8 bits) and Port 3 (upper 8 bits), is connected to a mounting rack with digital output modules. These outputs from the cell controller are conveyor and AS/RS command inputs to the PLC.

Software

Like the other workcell controllers, this cell software **mhlwcc** also affords the opportunity to run the system from the systems controller, or independent of it, from the workcell itself (Figure 9.2). During local control, the user can make the conveyor transport a pallet from one location to another (Figure 9.3), using macros developed which interface with the PLC program developed by Muralikrishnan et al. [14]. Each macro basically sends a conveyor task code (see Table 9.1) on the workcell digital output port 2 (bits 0 through 3), which are inputs to the PLC. The macros are named according to their source and destination. The macros developed are sttost(), sttoas(), sttomi(), sttoop(), astomi(), astoop(), astost(), mitoop(), mitoas(), mitost(), optost(), optoas(), optomi().

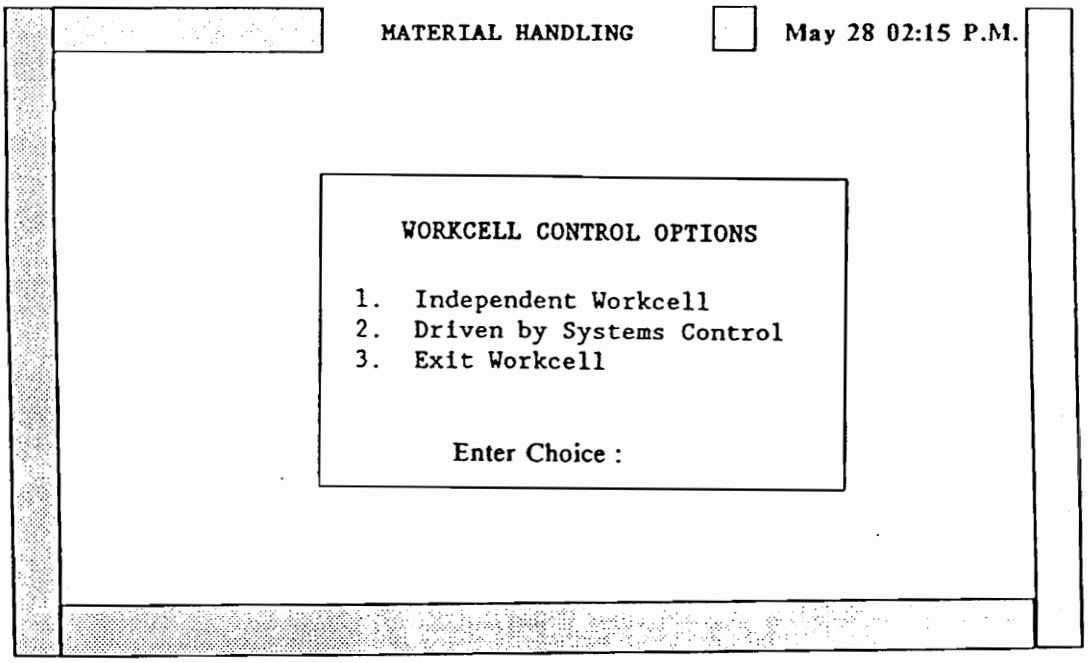


Figure 9.2 Material handling workcell control options

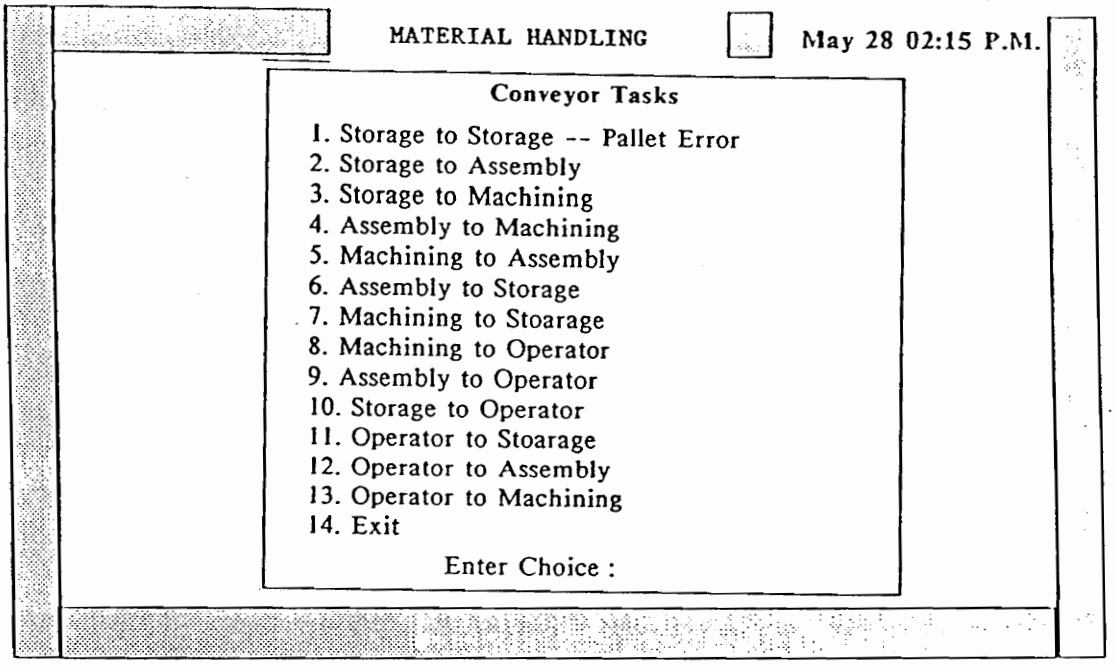


Figure 9.3 Conveyor task menu

As in the case of the other workcells, when the material handling workcell is driven by the systems controller, it executes macros of the same name as the command it receives from the controller (Table 9.1).

TABLE 9.1

System control commands to Material Handling

Command	Task	Code	
		Binary	Decimal
sttost	From storage to storage	0001	1
sttoas	From storage to assembly	0010	2
sttomi	From storage to machining	0011	3
sttoop	From storage to operator	0100	4
astomi	From assembly to machining	0101	5
astoop	From assembly to operator	0110	6
astost	From assembly to storage	0111	7
mitoop	From machining to operator	1000	8
mitoas	From machining to assembly	1001	9
mitost	From machining to storage	1010	10
optost	From operator to storage	1011	11
optoas	From operator to assembly	1100	12
optomi	From operator to machining	1101	13

10. SYSTEMS CONTROLLER

The systems controller is responsible for coordinating the activities of the different workcells. It is linked to the other workcell controllers only by STARLAN. It is not hardwired to any device and does not have any cell I/O of its own, unlike the other cell controllers. However the systems controller software **syscon** has been provided the ability to control the primitives of equipment in the 3 workcells under its domain. It also can execute all the macros of the workcells, described above, with status feedback (Figure 10.1). This control and status feedback is achieved by having a strong message system.

Each cell controller has associated with it two input and one output files. Table 10.1 summarizes the input and output files through which the systems controller and the workcell controllers communicate with each other. The command input files to the workcell controllers have an 'in.dat' ending, while the status output files from the workcell controllers which are read by the systems controller have an 'out.dat' ending. When the primitives of equipment of either the machining or assembly workcell are

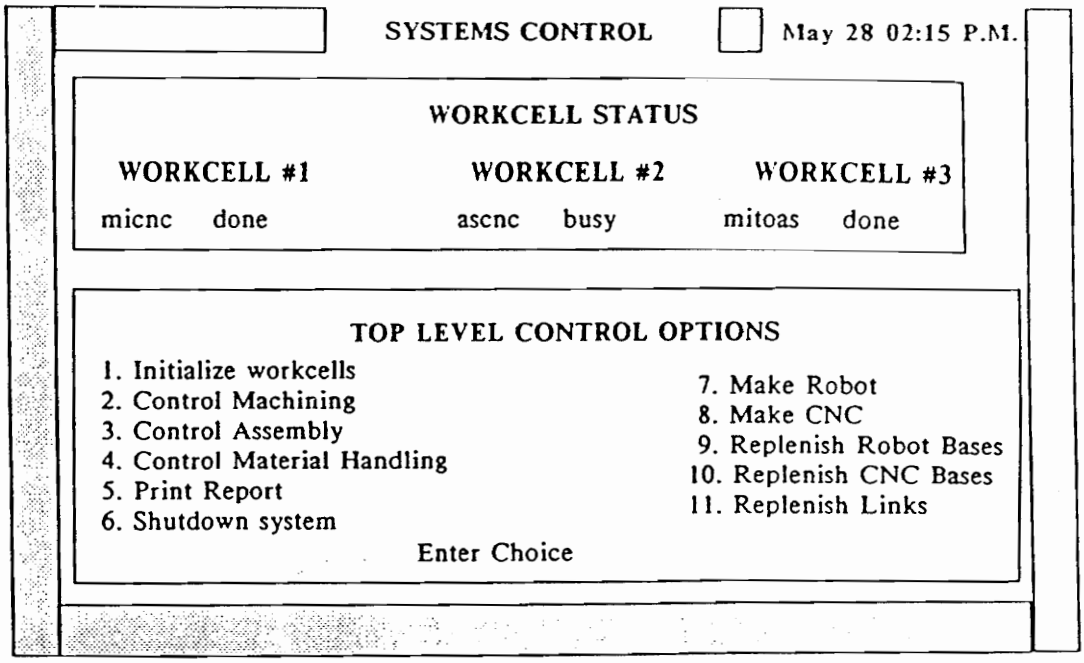


Figure 10.1 Systems control main menu

TABLE 10.1

MESSAGE FILES

Machining Workcell

mwcin.dat - Command input file to machining workcell

mwcout.dat - Status output file from machining workcell

mwc2in.dat - Primitives command input file to machining workcell

Assembly Workcell

awcin.dat - Command input file to assembly workcell

awcout.dat - Status output file from assembly workcell

awc2in.dat - Primitives command input file to assembly workcell

Material Handling

mhcin.dat - Command input file to material handling

mhcout.dat - Status output file from material handling

mhc2in.dat - Primitives command input file to material handling

Record File

history.dat - File in which the time and content of all messages passed in the above files, for a session, are appended

to be controlled from the systems controller, the message written into 'mwcin.dat' or 'awcin.dat' is "equipment", while the specific task and associated variables for the primitive control is written in the relevant workcell input file with filename ending '2in.dat'. In the former case the WCC uses the routine communicatein() to read the task command, while in the latter the WCC reads the specific task variables with the routine c2ommunicatein() developed in the toolkit file 'lan.h'.

All the primitives of the three workcells, except the read_io(), read_reject_status() and read_machine_status(), developed for the robots in the machining and assembly workcells, are implementable from systems control.

The systems controller has cell control "super-macros" to automate the whole FMS, by sequencing the macros of the different workcells, to achieve co-ordination between workcells (Table 10.2). There are 5 super-macros grouped into two sets of functions: Rbasreplenish(), cbasreplenish() and lnkreplenish() replenish raw material stock from storage to the part feeders. Makerobot() and makecnc() take an empty pallet through the kit building, machining, assembly and transportation processes necessary to make a robot or a CNC machine.

TABLE 10.2 Super-macros developed for Systems control

Name	Task
makerobot()	Makes robot - starts by bringing an empty pallet from storage to the assembly cell, builds a robot kit pallet, transports it to the machining cell, machines the parts, takes the finished part pallet back to the assembly cell where the parts are assembled. The assembled robot pallet is then taken to the operator.
makecnc()	Makes CNC - starts by bringing an empty pallet from storage to the assembly cell, builds a CNC kit pallet, transports it to the machining cell, machines the parts, takes the finished part pallet back to the assembly cell where the parts are assembled. The assembled CNC pallet is then taken to the operator.
rbasreplenish()	Brings a robot base raw material pallet from storage to the assembly cell, where the material is stored in a bracket. The empty pallet is returned to storage.
cbasreplenish()	Brings a CNC base raw material pallet from storage to the assembly cell, where the material is stored in a bracket. The empty pallet is returned to storage.
lnkreplenish()	Brings a link raw material pallet from storage to the assembly cell, where the material is stored in a bracket. The empty pallet is returned to storage.

For example, the makerobot() routine functions as follows:

1) Builds a robot kit on an empty pallet brought to the assembly workcell from storage, by the conveyor.
2) Transports the raw material kit pallet to the machining workcell for machining.
3) After machining, transports the machined parts pallet back to the assembly workcell for assembly.
4) Assembles the final product using the fixtures, and then transports the pallet to the operator interface.

Table 10.3 provides the exact parameter specifications of the above routine. Makecnc() is a similar routine to machine and assemble the CNC. Rbasreplenish(), cbasreplenish(), and lnkreplenish() are three routines to transport raw material from storage to replenish the brackets in the assembly workcell. The empty pallets are then sent back to storage.

The systems controller will also have the ability to generate a session report of the time when tasks were given to the different workcells, and when each task was completed. This is done by keeping track of the time and message in each of systems controller to cell controller message files, and appending this information to a file

TABLE 10.3 Breakdown of macro makerobot()

of the Systems controller

makerobot() functions as follows:

1) Sends command 'sttoas' to mhcin.dat
2) Waits for message 'sttoas done' from mhcout.dat
3) Sends 'burobot' to awcin.dat
4) Waits for 'burobot done' from awcout.dat
5) Sends 'astomi' to mhcin.dat
6) Waits for 'astomi done' from mhcout.dat
7) Sends 'mirobot' to mwcin.dat
8) Waits for 'mirobot done' from mwcout.dat
9) Sends 'mitoas' to mhcin.dat
10) Waits for 'mitoas done' from mhcout.dat
11) Sends 'asrobot' to awcin.dat
12) Waits for 'asrobot done' from awcout.dat
13) Sends 'astoop' to mhcin.dat
14) Waits for 'astoop done' from mhcout.dat

'history.dat', before erasing the message file. The function record() achieves this ability.

11. CONCLUSIONS & RECOMMENDATIONS

There were two objectives of this project:

1) to develop a test and debug capability for the communication links within an FMS, and
2) to demonstrate the direct applicability of primitive communication routines developed, to achieve high level operations for an integrated and automated system.

Both of the objectives were met successfully. A toolkit of functions was augmented and modified to enable control of robots, CNC machines, and workcell digital I/O using DATA TRANSLATION 2808 and DIO64 boards. The toolkit also includes functions for generic serial and LAN communications. For the serial communications aspect, routines have been developed which can be used to communicate over a RS-232C serial line, using any set of transmission parameters. For the LAN communications aspect, routines have been developed which enable message passing between computers. The developed toolkit functions insulate the user from lower level communications. The user can directly use the library functions to implement more complex

communications. The routines of the toolkit can be exercised by the user by using a menu driven program **toolkit.**

The toolkit functions that were developed were demonstrated on an FMS. The FMS was divided into three workcells. Each workcell was controlled by a workcell controller, itself under the supervision of a systems controller. The workcell controllers themselves were hardwired to equipment like robots, CNC machines and material handling equipment. The FMS was integrated as one unit, and made driveable from systems control. The systems controller was given the ability to control the primitives of the equipment of the different workcells, as well as control execution of a sequence of macros developed for the different workcells.

Currently the FMS has been implemented without using an AS/RS or a vision system. Future work can involve integration of these two modules into the system. Alongside, with the implementation of the AS/RS and vision system, it is also necessary to develop software to keep track of the inventory in the system.

The present material handling system software has been developed for a single pallet in the system. Future work can be directed towards expanding this to a 'multiple pallet on conveyor' system. This will require accounting for scheduling features.

The present system software assumes an ideal operation - the ability to detect errors and recover from them has been beyond the scope of this project. Future work can involve incorporation of a higher level AI controller, which schedules and prioritizes tasks, keeps track of inventory in a data base, and is able to intelligently recover from errors.

REFERENCES

1. Bidani, S., "The Applicability of APT towards meeting Control needs in Discrete Parts Manufacturing," IEOR MS Thesis, Virginia Polytechnic Institute and State University, Sept 1989.

2. CIM Laboratory Cable Documentation, March 1989

3. Data Translation Inc., "Personal Computer series user manual for DT 2808 single board analog and digital I/O system," Data Acquisition and Control Products, 1983.

4. Dyna Electronics product manual, "DM 2400/2200 Programming manual."

5. Dutton, D.J., "The Design and Implementation of a Cell Controller," M. S. Thesis, Department of Industrial and Systems Engineering, Georgia Institute of Technology, 1987.

6. Economy, T. et al., "Assembly Workcell Report," IEOR 5324 Class Project, Spring 1990.

7. Ekong, Etim S., "Controls : The Bridge to Systems Integration," 16th Annual International Programmable Controllers Conference and Exposition, Detroit, Michigan, April 7-9, 1987.

8. Grierson, D. K., "Factory Automated Systems - Solutions through integration," Autofact 5, MS83-731, 1983.

9. Guleri, A., "Device Driver Development and Implementation for Workcell Control," IEOR M.E. Project, Virginia Polytechnic Institute and State University, Dec 1988.

10. IBM Manufacturing System Software Library, "Aml/Entry Version 4 User's Guide," Second edition, August 1988.

11. Industrial Computer Source product manual, "Model DIO64 reference manual," 1988.

12. Industrial Computer Source product manual, "Model Commbios reference manual," 1988.

13. Jones, A. T., and McLean C. R., " A Proposed Hierarchical Control Model for Automated Manufacturing Systems," Journal of Manufacturing Systems, Vol.5, No.1, November 1987.

14. Mechler, Bob, "Cell Controller's Evolution to a Product," 16th Annual International Programmable Controllers Conference and Exposition, Detroit, U.S.A., April 7-9, 1987.

15. Muralikrishnan C.K. et al., "Systems Control and

Material Handling Report", IEOR 5324 Class Project, Spring 1990.

16. Nair, G., et al., "Assembly Workcell Controller Report," IEOR 5324 Class Project, Spring 1989.

17. Quatse, Jesse T., "Requirements for Real Time Cell Control," 15th Annual International Programmable Controllers Conference and Exposition, Detroit, U.S.A., April 8-10, 1986.

18. Ranky, Paul G., Computer Integrated Manufacturing, Prentice Hall International, UK., 1986.

19. Ridgway, A., et al., "Material Handling Workcell Controller Report," IEOR 5324 Class Project, Spring 1989.

20. Romano, K., et al., "Machining Workcell Controller Report," IEOR 5324 Class Project, Spring 1989.

21. Saboo, J. V., et al., "Systems Controller Report," IEOR 5324 Class Project, Spring 1989.

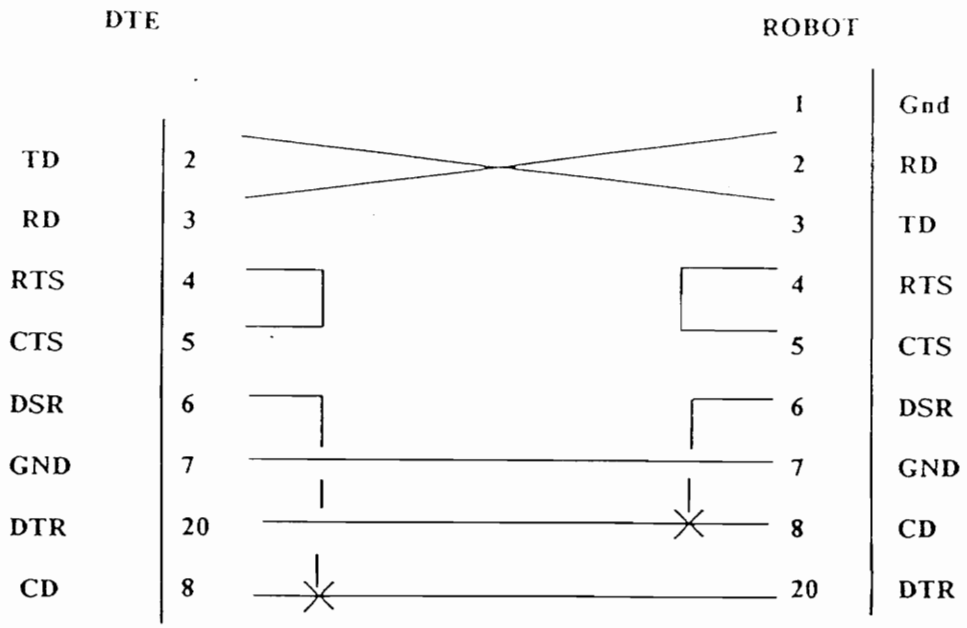
22. Wiegmann D. et al., "Machining Workcell Report," IEOR 5324 Class Project, Spring 1990.

23. Zhang, Y. L., "An Integrated Intelligent Shop Control System," IEOR M.S. Thesis, Virginia Polytechnic Institute and State University, April 1989.

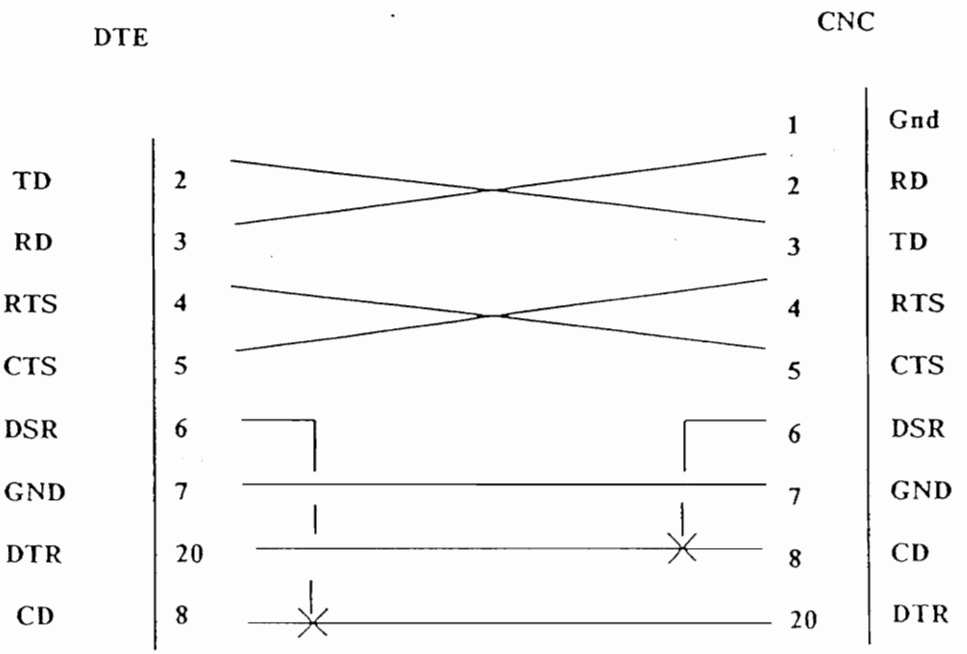
APPENDIX A

Procedure to load multiplexer device driver

The Industrial Computer Source software for the multiplexer device driver, has in its commbios subdirectory, an executable program called "install". Run this program and respond appropriately to each question [12]. This creates a file "commbios.com" on the current directory. Include the executable file "commbios" in the "autoexec.bat" file, so that the multiplexer device driver is automatically loaded whenever the computer is booted up.



Local RS-232 cable wiring for IBM 7545/7 robot

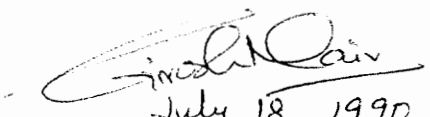


RS-232 cable wiring for DYNA CNC machines

APPENDIX B

VITA

Girish Nair was born on December 14th, 1965 in Bombay, India. He did his schooling at The Cathedral & John Connon School, Bombay, India, and then at St Xavier's College, Bombay, India. He graduated in May 1988 with a Bachelor of Technology degree in Mechanical Engineering from the Indian Institute of Technology, Bombay, India. He then enrolled in the Department of Industrial Engineering and Operations Research at Virginia Polytechnic Institute and State University, Blacksburg, Virginia, USA for a Master of Engineering degree.



July 18, 1990

Girish Nair