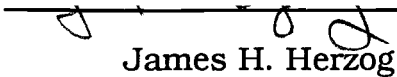


## **AN ABSTRACT OF THE THESIS OF**

Choo-Chiang, Lim for the degree of Master of Science in Electrical and Computer Engineering presented on December 8, 1989.

Title : A Programmable System Scheduler For Control Oriented Local Area Networks (COLAN)

Abstract approved: Redacted for privacy  
  
James H. Herzog

The Control-Oriented Local Area Network (COLAN) is a distributed control system for a series of networked microcontrollers, which has been under development at Oregon State University since 1986. A reliable master controller, functioning both as a task scheduler and as a network controller, is required to allow users to perform task programming, the transmission and reception of commands and data packets, and network status monitoring.

The Task Master Controller (TMC) has been designed and developed to provide these capabilities. The TMC provides an integrated environment to aid in the editing of task programs, the execution of task programs, the interpretation of program statements, the manipulation of files, the maintenance of a communication protocol between the host computer and remote

microcontrollers, the maintenance of a device and task library, and the display of network status. The TMC provides two different modes of operation, a user mode and a command mode, to allow both the novice user and the experienced system developer to use the system.

The TMC language also includes such basic programming language elements as conditional statements, repetitive statements, and block statements. It also includes such built-in functions as time delay, print message, reception of data, and save response. These elements provide the skilled system developer with a powerful tool to program tasks in any desired sequence. It provides the novice user with a friendly user interface to schedule tasks by selecting from a menu of high level commands included in the system library.

A Programmable System Scheduler  
For Control Oriented Local Area Networks (COLAN)

by

Choo-Chiang, Lim

A THESIS  
Submitted to  
Oregon State University

in partial fulfillment of  
the requirements for the  
degree of  
Master of Science

Completed December 8, 1989  
Commencement June, 1990

APPROVED :

Redacted for privacy

Associate Professor of Electrical and Computer Engineering in  
charge of major

Redacted for privacy

Head of Department of Electrical and Computer Engineering

Redacted for privacy

Dean of Graduate School

Date thesis is presented December 8, 1989

## **ACKNOWLEDGEMENT**

I would like to express my sincere gratitude to my advisor, Prof. James Herzog, for his guidance, encouragement and support throughout this study and for the many hours he spent reading my work and commenting on it while it was being done. I also like to thanks Prof. Joel Davis, Prof. Roy Rathja and Prof. Bella Bose for acting as my committee members and for giving me useful advice.

I warmly thank all of my friends and colleagues whose assistance, encouragements have been invaluable.

Finally I thank my parents and family members for their financial support and encouragement throughout my education.

## TABLE OF CONTENTS

<u>Chapter</u>	<u>Page</u>
1. INTRODUCTION -----	1
1.1 Motivation -----	1
1.2 Background -----	1
1.3 Multiple Microprocessor Systems -----	2
1.4 Local Area Networks -----	4
1.5 TASTMASTER Systems -----	5
1.6 Control-Oriented Local Area Networks -----	7
1.7 Task Scheduling Problem for COLAN -----	8
1.8 Objective of the Thesis -----	9
1.9 Outline of the Thesis -----	11
 2. DATA COMMUNICATION INTERFACE -----	 15
2.1 Overview -----	15
2.2 Host Computer -----	15
2.3 Microcontroller System -----	16
2.4 EIA RS-485 Bus Standard -----	17
2.5 RS-232C/RS485 Converter -----	18
 3. DESIGN AND IMPLEMENTATION OF TMC SOFTWARE -----	 22
3.1 Overview -----	22
3.2 Main Program -----	23
3.3 Communication Protocol -----	23
3.4 Interpreter -----	26
3.5 Task Processor -----	27
3.6 Screen Editor -----	30
3.7 System Library -----	31
3.8 File Handling -----	32
3.9 Network Monitoring -----	33
3.10 User Interface -----	33
3.11 Miscellaneous -----	34

## TABLE OF CONTENTS (continued)

<u>Chapter</u>	<u>Page</u>
4. USER MANUAL FOR TMC SOFTWARE-----	40
4.1 Overview-----	40
4.2 Installation-----	40
4.3 Getting Started-----	41
4.4 Using the Menu System-----	41
4.4.1 Main Menu-----	42
4.4.2 Quick-Ref Line-----	43
4.4.3 Editor Window-----	44
4.4.3.1 User Mode Task Editing-----	44
4.4.3.2 Command Mode Task Editing-----	46
4.5 Menu Commands-----	46
4.5.1 File Menu-----	46
4.5.2 Edit Command-----	48
4.5.3 Execute Menu-----	48
4.5.4 Setup Menu-----	50
4.5.5 Library Menu-----	51
4.5.6 NetInfo Command-----	52
4.6 Basic Task Program Statements-----	52
4.6.1 Command Package Statements-----	52
4.6.1.1 Device Address-----	53
4.6.1.2 Prefix-----	53
4.6.1.3 Task Number-----	54
4.6.1.4 Suffix-----	55
4.6.1.5 Data Field-----	56
4.6.1.6 Echo Requests-----	56
4.6.2 Data Packet Statements-----	57
4.6.3 Conditional Statements-----	57
4.6.4 Repetitive Statements-----	58
4.6.5 Block Statements-----	58
4.6.6 Built-In Function Statements-----	59

## TABLE OF CONTENTS (continued)

<u>Chapter</u>	<u>Page</u>
5. SUMMARY AND RECOMMENDATIONS -----	67
5.1 Summary -----	67
5.2 Recommendations for Future Research -----	68
 BIBLIOGRAPHY -----	 71
 APPENDIX A TMC APPLICATION EXAMPLES -----	 73
A.1 Application Overview -----	73
A.2 Example 1: A Museum Project -----	74
A.3 Example 2: A Greenhouses Temperature Control Project -----	 76



## LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
1-1 Tightly Coupled Multiprocessor System -----	12
1-2 Loosely Coupled Multiprocessor System -----	12
1-3 System Configuration For TASKMASTER-----	13
1-4 Network Configuration For COLAN-----	13
1-5 Robot Arm Using COLAN Configuration -----	14
2-1 Half-duplex RS-485 Multidrop Network-----	20
2-2 RS-232C/RS-485 Converter-----	21
3-1 Flowchart for Establish Asynchronous Communication-	36
3-2 Circular Queue Operations-----	37
3-3 Syntax Graph For A TMC Statement-----	38
3-4 Data Structure For TMC Library-----	39
4-1 The Main Menu-----	62
4-2 User Mode Task Editing-----	62
4-3 Command Mode Task Editing-----	63
4-4 The File Menu-----	63
4-5 The Execute Menu -----	64
4-6 The Setup Menu -----	64
4-7 Device Library-----	65
4-8 Task Library -----	65
4-9 Network Status Window-----	66

# **A PROGRAMMABLE SYSTEM SCHEDULER FOR CONTROL ORIENTED LOCAL AREA NETWORKS (COLAN)**

## **CHAPTER 1**

### **INTRODUCTION**

#### **1.1 Motivation**

The purpose of this research was to develop and implement a reliable software for the host computer in COLAN (Control Oriented Local Area Network) to allow users to synchronize and control all the system activities and monitor the network status.

#### **1.2 Background**

The development of VLSI technology during the past decade has pushed the evolution of the microprocessor to its practical limits. Methods used to increase microprocessor performance include increased word (data-bit) length, increased clock speed, and the use of more sophisticated architectures. However, none of these factors will continue to increase at the rate experienced during the past decade. The current 16-bit and 32-bit

microprocessors with clock rates of 16 - 25 MHz have achieved a much greater performance than 8-bit microprocessors operating at clock rates of 5 - 12 MHz. However, word lengths of 64-bits or more operating at speeds beyond 25 MHz greatly complicate system design. Sophisticated architecture, such as pipelining and instruction caches, can be used to increase performance, but only within restricted limits. The one approach that has not reached its practical limits is the use of multiple microprocessors.

Multiple microprocessor systems can be used to provide an appropriate solution to the demand for additional computing power to meet new requirements. They can support complex applications, ranging from instruments and dedicated control systems to large supercomputers.

### **1.3 Multiple Microprocessor Systems**

Multiple microprocessor systems can be categorized by the degree of coupling and the nature of the intercommunication between processors. Coupling refers to the ability of the various elements to share resources, with the two extremes being tightly coupled and loosely coupled systems [ FATH 83].

Tightly coupled systems are also known as multiprocessor systems. Figure 1-1 shows a tightly coupled system, in which the processors share a common primary memory and bus structure. A single common operating system is used to control and coordinate all interactions between processors and processes. I/O facilities

and other system resources are generally shared among the processors. However, some resources may be dedicated to specific processors. Synchronization between cooperating processors is needed in tightly coupled system. The major limitation of a tightly coupled multiple processor organization is the possibility of primary memory access conflicts. This memory-processor bottleneck tends to put an upper bound on the number of processors which can be supported by a single operating system. Tightly coupled systems must be very carefully designed to use more than a few processors effectively.

Loosely coupled multiple processor organizations contain a number of independent processors that can be geographically dispersed. This type of organization is also known as a computer network. Figure 1-2 shows a loosely coupled system, in which each processor has its own local resources and can therefore operate independently of the other processors. Each processor executes its own program from its local memory. The various processors in the system are interconnected via a communication interface. The communication links are generally high-speed serial lines. Interprocessor communications follow a rigid communication protocol. Many multiple microprocessor applications are loosely coupled systems. Relatively independent microprocessors, each dedicated to a particular task, operate simultaneously and communicate only when necessary. This breaks the memory-processor bottleneck, since each processor has its own local memory and operating system.

## 1.4 Local Area Networks

A local area network (LAN) is a loosely coupled multiple processor organization. Since 1980, LANs have been used to interconnect computers, terminals, workstations, data base servers, and other peripheral devices within a building or local group of buildings. Some of the basic characteristics of a LAN include [COTT 1980]:

- A high rate of data exchange (0.1 to 100 Mbps);
- A high degree of data transfer accuracy (less than one undetected error per  $10^{12}$  bits ~ 1 trillion characters);
- Low hardware cost (less than \$1,000 per network connection, with the anticipation of future declines as logic costs continue to decline);
- High network reliability/availability through the use of state-of-the-art transmission technology (i.e., a failure results in the disconnect of one user or microcomputer, but without damaging the network); and
- Geographically local structure (a range of 300 - 10000 ft).

As factories and offices move toward automation, the use of LANs for sharing information and computer-related resources is expected to increase in like proportion.

## 1.5 TASKMASTER Systems

The TASKMASTER system is a task-driven distributed microcontroller system developed at Oregon State University. The system was created to test the idea of a simple real-time control system utilizing multiple microcontrollers controlled by a single scheduler [HERZ 87].

The original TASKMASTER system was organized with a host computer (PC) as the system scheduler. Individual microcontroller based boards, called TASKMASTERS, link to the host and each other in a daisy-chain structure (Figure 1-3). The host computer is used to schedule, control, synchronize and coordinate all the system activities by sending a command packet to each of the TASKMASTER units as appropriate. These activities are carried out by the TASKMASTER units using their specialized application interfaces. At the request of system scheduler, data collected by TASKMASTER units can also be transferred back to host computer. In other words, the host computer serves as the system's master controller for a series of slave microcontrollers. The TASKMASTER units are strategically located to maximize equipment availability and to provide for graceful system degradation should the system fail.

Commands to the TASKMASTER units are in a "task" format. The tasks are commands that utilize the resources of the microcontroller to perform a specific action. The resources are the various pieces of interface hardware used in the specialized control and monitoring activity. Each task is a preprogrammed subroutine

written specifically to accomplish a specific action in the context of the control problem. The various tasks available to the system are permanently stored in the local program memory of the individual microcontrollers. A typical task command packet usually consists of the following information:

- (1) Address : Specifies the destination microcontroller.
- (2) Prefix : Specifies the methods or conditions under which the task is to be started. A task may be added to the local task queue, executed immediately or execution may be delayed until the arrival of a "sync" command from the host.
- (3) Task Number : Identifies the specific task to be performed.
- (4) Postfix : Specifies the methods or conditions under which the task is to be terminated. Tasks may be discarded after execution or may be requeued for repeat running.
- (5) Parameters : A task may require specialized data or further specification before it can run.

Two special characters, "{" and "}", are used to indicate the start and the end of a packet. By the proper sequencing of tasks, complex control and monitoring activities can be performed [HERZ 87].

## 1.6 Control-Oriented Local Area Networks

The TASKMASTER system was modified to change its interconnection structure from the daisy-chain to that of a LAN. These modified systems, called COLAN, for Control Oriented LAN, were designed to bring high performance control networks within such structures as factories, school or offices.

The first COLAN was developed in 1986 by Y. P. Zheng [ZHEN 86]. Since that time five different types of COLANs have been implemented, including COLAN II [KAO 87], COLAN III [EUM 87], COLAN IV [THYE 88], and COLAN V [KUMA 89]. Each of these systems utilizes a different approach to the implementation of a network.

The basic COLAN configuration (Figure 1-4) consists of several microcontrollers and a host computer linked to a local area network by a RS-485 bus. The bus topology provides a simple but effective way of realizing a decentralized control structure of a LAN. The RS-485 bus standard is adopted to achieve a high performance with a low cost twisted-pair cable. It supports data rates up to 10 Mbps with a 50 ft cable, or 100 Kbps with a 4000 ft cable. At a data rate of 9600 baud, the RS-485 bus of COLAN can operate with a cable length up to 4000 ft.

The host computer and the microcontrollers are connected to the RS-485 bus through an RS-232C/RS-485 converter which is used to change the RS-232C communication standard to RS-485 communication standard, or vice versa. RS-232C is a commonly used asynchronous serial data communication standard and, is



supported by both the host computer (PC) and microcontrollers used in COLAN. However, the maximum cable length supported by RS-232C is limited to 50 ft. This is not sufficient for many real-time distributed control applications. RS-485 communications standard, on the other hand, allows for a longer cable length. The RS-232C/RS-485 converter provides an interface between the host computer and microcontroller and the RS-485 bus in order to have a longer data communication range.

### **1.7 Task Scheduling Problem for COLAN**

From the development of the initial COLAN to the implementation of COLAN V, system design has been concentrated on the establishment of a communication standard and protocol method, as well as on the development of a microcontroller operating system. The principal function of the host computer program is limited to sending and receiving commands and data packets to and from the microcontrollers on the network. The host programs generally consists of menu driven routines that allow the user to select a specific microcontroller (called NIUs or Network Interface Units [THYE 88]) and the task that it is to be executed. None of these programs, however, provide what was originally intended for the TASKMASTER system: A system scheduler that provided control and synchronization for all of the connected microcontrollers. While these programs are sufficient to test the network reliability, they are too primitive for real-time,

practical applications. Therefore, in order for COLAN to be used in real-time applications, the primary need was to develop a task scheduler to synchronize the system activities and handle flow control of communication packets.

For example, a robotic arm system, as shown in Figure 1-5, utilizes a microcontrollers at each of three joints, Each monitors a position sensor to provide closed-loop control. In addition, the joint processor can perform some of the computation necessary to execute a motion command. A central scheduler is required to handle the user interface and gives directions to each of the joint processors by sending sequences of task commands to all the joint processors concurrently. The design and implementation of a task scheduling software will be the main topic of this thesis.

## **1.8 Objective of the Thesis**

The objective of this project is to design the Task Master Controller (TMC), a programmable task scheduling software which has the functions and capabilities discussed in the previous section. More specifically, The TMC includes the following features :

- **Communication protocol:** A communication protocol routine to maintain reliable communication sessions between the host computer and the slave microcontrollers in the network.
- **Editor:** An on-screen editor for task programming

(scheduling). A task program is a sequence of task commands for the network microcontrollers.

- File manipulation functions: File manipulation functions including loading files, creating new file, or saving edited files, are made available to the user.
- Interpreter: A interpreter is invoked to check the syntax of task program and assure system compatibility.
- Task processor: A task processor is needed to execute the task program. Task programs are executed in sequential order in two basic ways: run and single step.
- Flow control statements: Statements such as *if/then/else* and *repeat* statements are provided for program flow control.
- Built in functions: Built in functions such as WAIT for response, Time DELAY, and PRINT message are provided to enhance performance of task program.
- User interface: TMC provide a friendly menu-driven style of user interface. All operations are activated by a press of a key.
- Network Monitoring: At any time, a user can obtain a list of active microcontrollers which can be accessed through the TMC.

The TMC also provides two different modes of operation for different categories of users. The command mode is designed for such expert users such as system developers. In this mode, tasks are entered directly from the keyboard in a predefined command-

packet statement format. The user mode is designed for general or novice users. In this mode, task programming is accomplished by the selection of a series of desired tasks from a predefined list (by the system developer) in the task library.

All responses received by the TMC from the slave microcontrollers are displayed immediately on the response window. Users can then choose to save all or part of the responses into a file for subsequent analysis.

The TMC offers network users a friendly environment to perform the task programming process. In the hands of an expert programmer, the TMC can be used as an extremely effective development tool.

## **1.9 Outline of the Thesis**

Following the introduction to the COLAN system and the uses of the TMC discussed in this chapter, the hardware implementation for the host computer and the system communication interface are considered in Chapter 2. Chapter 3 includes an analysis of the design and implementation of the TMC program. More detailed TMC operations are considered in Chapter 4, which may also be used as a TMC user manual. The final chapter provides a brief summary of the findings of this study and provides suggestions for further development. Examples of useful applications are given in Appendix A.

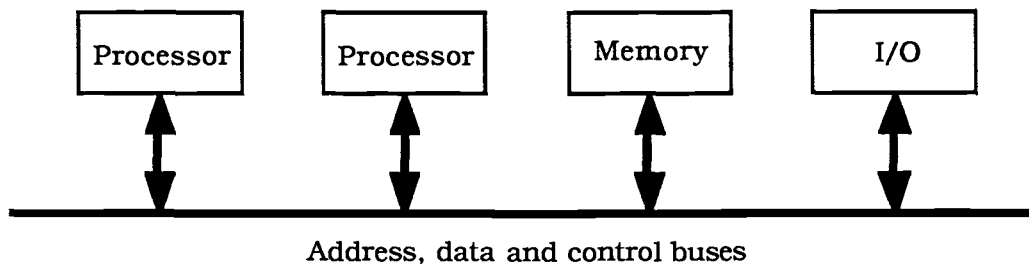


Figure 1-1. Tightly Coupled Multiprocessor System.

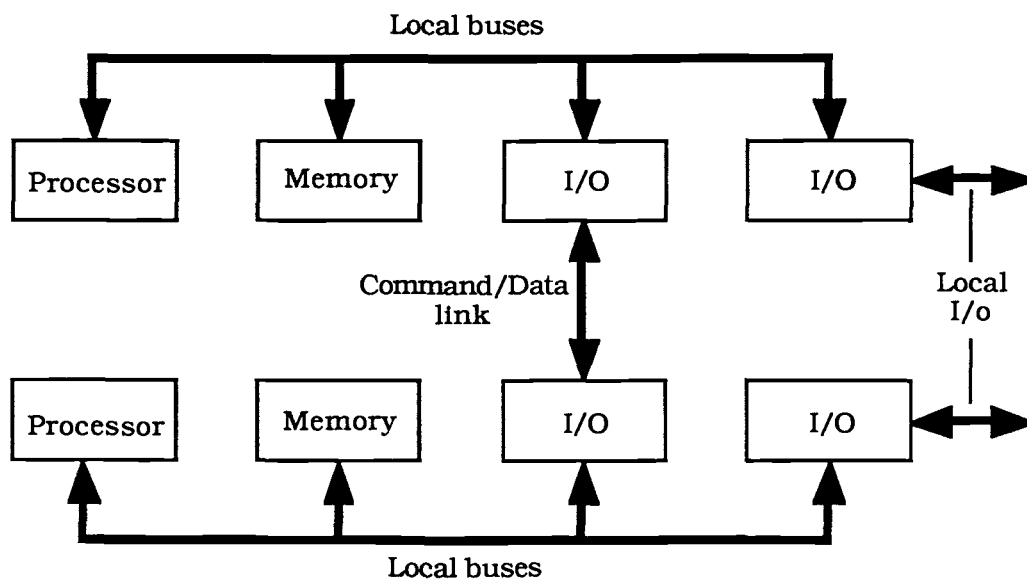


Figure 1-2. Loosely Coupled Multiprocessor System.

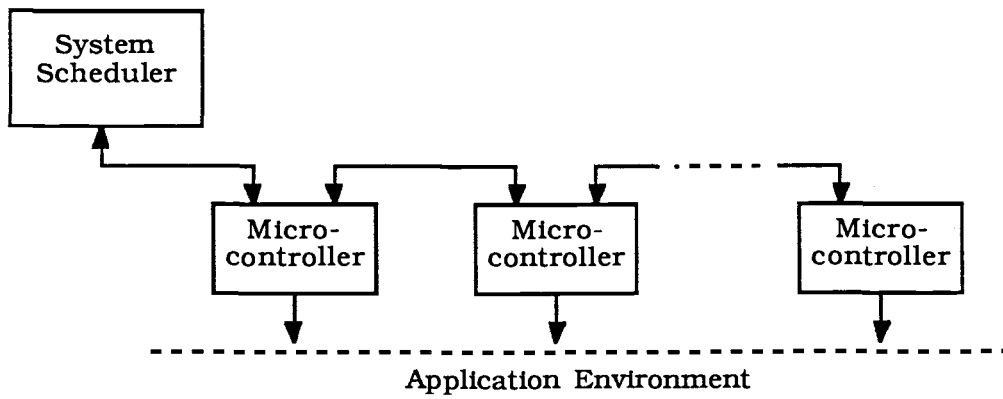


Figure 1-3. System Configuration for TASKMASTER

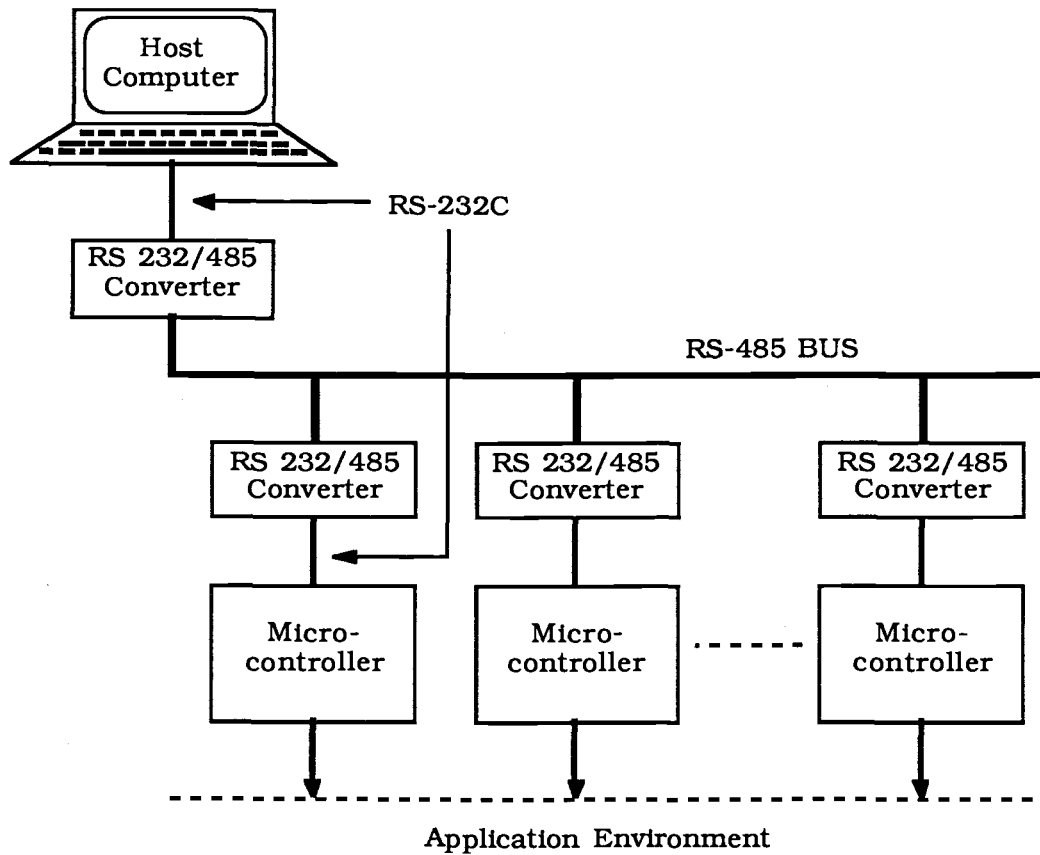


Figure 1-4. Network Configuration for COLAN

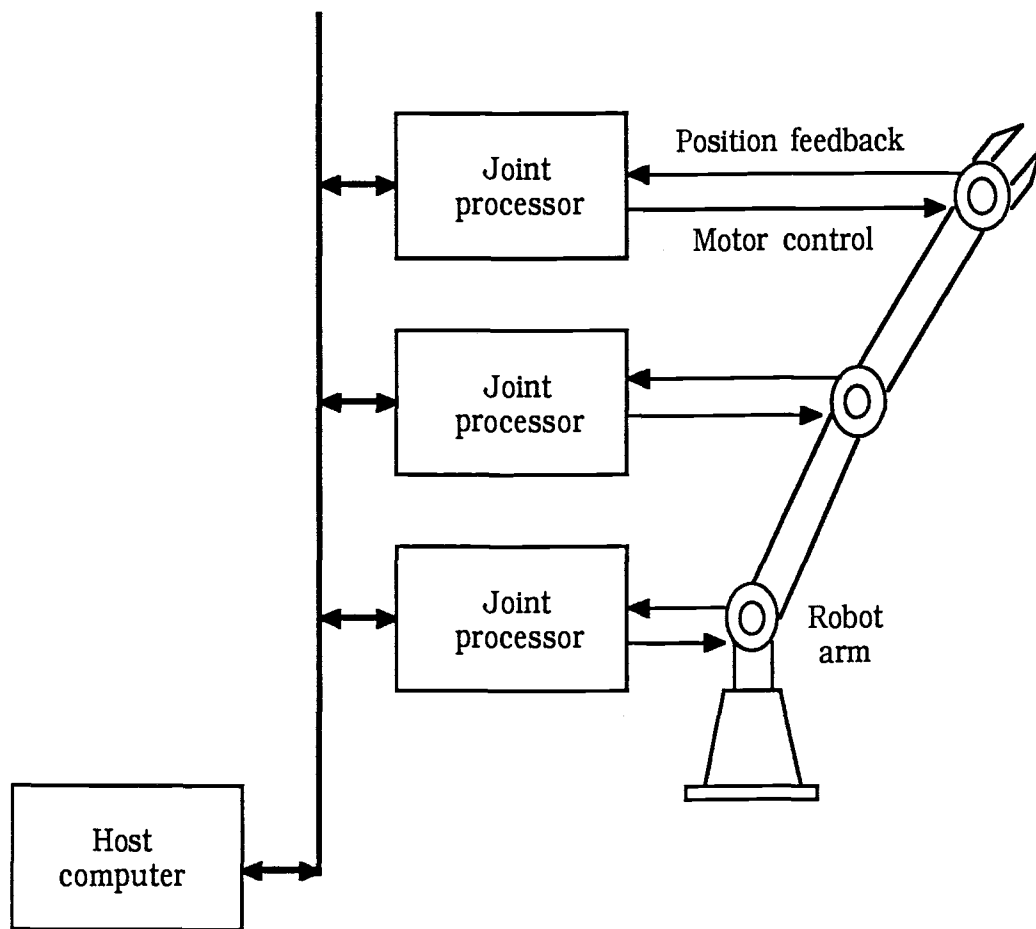


Figure 1-5. Robot Arm Using COLAN Configuration.

## **CHAPTER 2**

### **DATA COMMUNICATION INTERFACE**

#### **2.1 Overview**

The hardware of COLAN consists of a number of nodes linked into a local area network by a twisted-pair cable. The hardware of each node consists of two components, a processor (host computer or microcontroller) and the RS-232C/RS-485 converter, which is used to provide the physical level data communication interface between the processor and the RS-485 communication bus. The following sections describe the implementation of a data communication interface based upon a simple RS-232C/RS-485 converter.

#### **2.2 Host Computer**

The data communication standard supported by the host computer is based upon an RS-232C serial interface. A standard IBM PC serial communication port, either COM1 or COM2, is used to provide half-duplex, asynchronous serial communications at a data rates ranging from 300 to 9600 baud. No additional interface



boards or other hardware modifications are required for the PC host.

### **2.3 Microcontroller System**

Two different types of 8-bit microcontroller, the Intel 8051 and the Motorola 68CH11, have been successfully used on the COLAN. Each of these microcontrollers has a self-contained asynchronous serial data port which is used for data communications. Since all communication in the COLAN uses the conventional asynchronous serial protocol, any microcontroller or microprocessor which supports a serial port can be used.

A microcontroller based system, or TASKMASTER unit, is used to provide both the data communication interface and specialized application interface. In addition to the CPU chip, the microcontroller based system includes ROM, RAM and I/O chips. Additional hardware requirements are strongly application dependent.

The simple serial interface circuit consists of a RS-232C line driver (1488) and line receiver (1489) pair, and a DB-25 connector. In some cases, a specialized serial communication coprocessor can be added to perform the serial communication.

## 2.4 EIA RS-485 Bus Standard

The RS-232C standard is the most widely used method for asynchronous serial data communications of 50 feet or less and is compatible with most computers, including the IBM PC. However, for many real-time, distributed control system applications, communications over longer cable lengths are required. The RS-485 is the EIA standard recommended for long distance asynchronous serial data communication.

The RS-485 bus uses balanced, or differential, drivers and receivers, each of which requires two wires per signal. This approach offers several advantages. Noise induced in the cable by electromagnetic interference affects both wires equally and the logic states represented by the differential voltage is not affected as strongly as in RS-232. Because of this immunity, high voltage swings between logic 1 and logic 0 are not needed for noise immunity as in the RS-232. Small voltage swings mean less cross talk between adjacent wires, thus longer cable can be used. Smaller voltage swings can also be made quickly, allowing the use of higher data rates.

When a 24-gauge twisted-pair cable (i.e., standard telephone cable) is used, with a capacitance of 16 pF/ft and a 100  $\Omega$  termination resistor at the receiver, the RS-485 bus supports a data rate of up to 10 Mbps for a 50 ft cable and up to 100 kbps for a 4,000 ft cable. Up to 32 drivers and 32 receivers can be connected to the system through a bidirectional bus. Figure 2-1 illustrates a half-duplex RS-485 multidrop network.

## 2.5 RS-232C/RS-485 Converter

The RS-232/RS-485 converter provides the interface between the host computer or microcontroller and the system bus by converting the PC's RS-232C communication signals to the RS-485 communication signals used in the COLAN system bus. The RS-232C/RS-485 converter consists of a differential line receiver (75175), a differential line driver (75174), an inverter (7404), and RS-232C drivers/receivers (MAX232). Figure 2-2 indicates the RS-232C/RS-485 converter circuit diagram.

The differential line receiver and driver are tri-state gates; when disabled, the output is in a high-impedance state. This allows the use of these drivers in systems with multiple transmitters. The line drivers have two outputs, forming a differential signal pair. The input is the data transmitted from the RS-232C.

The line receiver has exactly the opposite configuration, with two differential inputs and an output of data compatible with the RS-232C standard. The tri-state control for the line driver is driven by a control signal from the microcontroller. In the case of host computer, the control signal is the request-to-send (RTS) signal, an RS-232 control signal provided through the PC serial port. An inverter is used to provide a complementary control signal for the line receiver. When the line driver is enabled for data transmission, the line receiver is disabled, and vice-versa. This prevents the unnecessary loop-back of the transmitted data through the line receiver. In the default mode the line receiver is enabled until data transmissions are required.

It is important to note that all RS-232C signals commonly operate with signal levels of +12 volts and -12 volts, and the line driver and receiver operate on 0 to +5 volt signal levels (TTL signals). This incompatibility in signal levels requires the use of additional RS-232C drivers and receivers (MAX 232) to convert RS-232C signals to the TTL signal level, and vice versa.

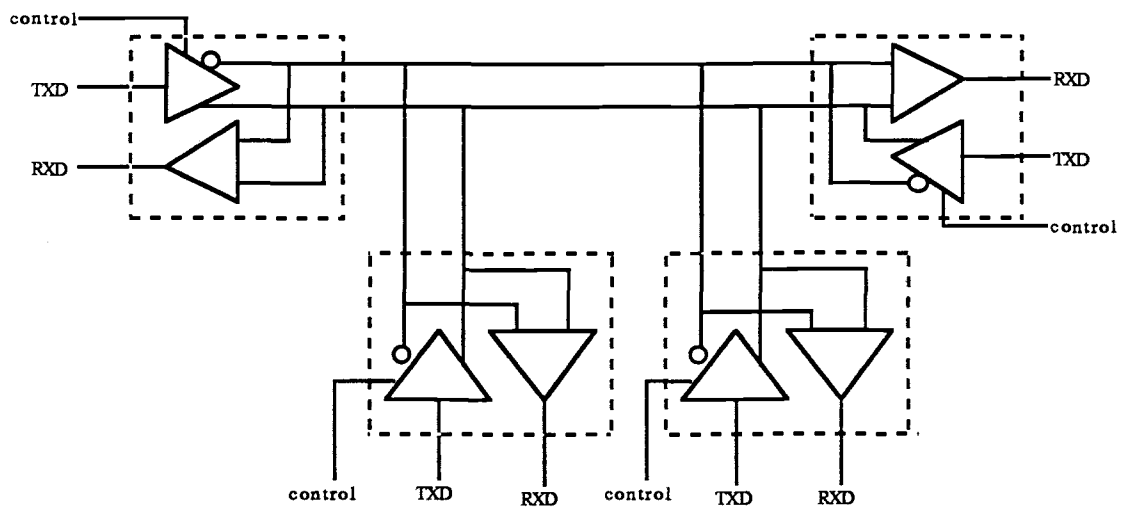


Figure 2-1. Half-duplex RS-485 Multidrop Network

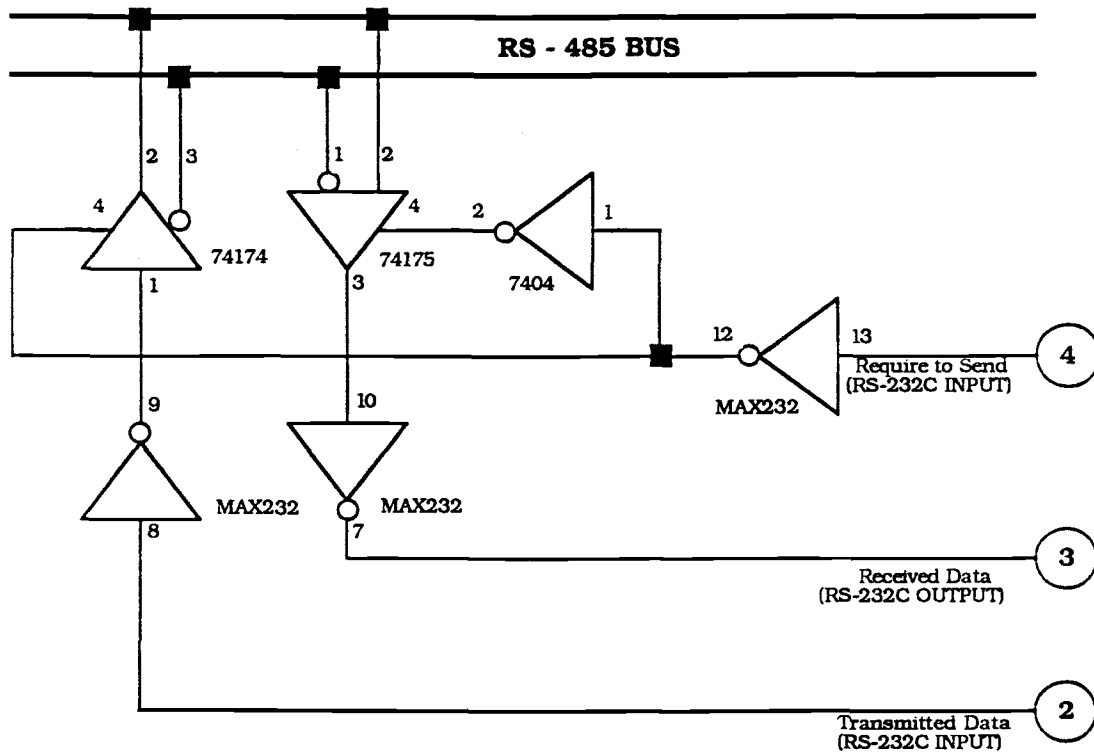


Figure 2-2. RS-232C/RS-485 Converter

## **CHAPTER 3**

### **DESIGN AND IMPLEMENTATION OF TMC SOFTWARE**

#### **3.1 Overview**

The TMC Task Scheduler program is written in Turbo Pascal, Version 5.0. The program is divided into 10 logical modules:

- main program,
- communication protocol,
- interpreter,
- task processor,
- screen editor,
- system library,
- file handling,
- network monitoring,
- user interface, and
- miscellaneous.

The design and implementation of each of the above modules is discussed in the following sections.

### **3.2 Main Program**

The main program initializes all the variables, including such global variables as the communication parameters, the monitor type (MGA, CGA, or VGA), foreground and background colors, and menu variables, as well as the system library. Control is then passed to the main menu, consisting of such selections as file handling, task program editing, task program execution, environment setup, the update of devices and the task library, or network status monitoring. In most cases, selection leads to a pull-down menu or subsequent pop-up menus with a number of other options. Depending on the user's choice, a series of functions or procedures are called to carry out the selected command.

### **3.3 Communication Protocol**

The most important part of the TMC program is the communication protocol, which is used to establish the communications between the PC and the microcontrollers attached to the network. The communication protocol routines are used to control the RS-232 serial port, the INS8250 universal asynchronous receiver-transmitter (UART), the 8259 interrupt controller, and the RS-232C/RS-485 converter. In addition, these routines specify baud rate, parity, stop bits, data bits, and serial port selection.



The communication protocol uses interrupt 12 [NORT 85] to provide simple telecommunications capability. The RS-232C/RS-485 converter is used to connect the host computer serial port to the network. The main program block calls several procedures and functions, outlining the steps necessary to establish asynchronous communications. These steps are shown on the flowchart in Figure 3-1. The following paragraphs describe the detail functions of each procedure.

### ***Set\_serial\_port***

This procedure uses the BIOS interrupt 14h to set the serial port to the parameters defined by the user. In this procedure, register AH is set to 0, which tells the BIOS to initialize a serial port. Register AL is set to a parameter byte whose bits contain the communication settings. Register DX is set to 0 for COM1 or 1 for COM2.

### ***Enable\_port***

The interrupt is installed by procedure *Enable\_port*. This procedure saves the old interrupt vector address, installs the new interrupt vector (address) of the interrupt handler routine *AsyncInt*. Whenever an interrupt is caused by an incoming character received from the slave microcontrollers in the network, *AsyncInt* is called to store the character in a circular buffer. *Enable\_port* also prepares the INS8250 UART chip for communications.

### ***Send\_packet or Receive\_packet***

After the serial port is initialized and the interrupt is installed, control is passed to either the procedure *send\_packet* or *receive\_packet*. The first transmits the task command packet to one of the distributed units character-by-character and the second receives a data packet from the network character-by-character. Prior to transmitting the packet, the line transmitter on the RS-232C/RS-485 converter board is enabled by an RTS signal generated by the *send\_packet* procedure. All received packets are displayed in the response window.

### ***Disable\_port***

In the final step in the communication procedure, *disable\_port* is used to install the original interrupt in the interrupt vector table and to reset the UART chip.

A circular input buffer is one of the central elements of an interrupt-driven communications program. Because data can arrive at the serial port any time, the interrupt handler must be able to capture and process that data while the computer is processing other functions. If the interrupt fails to store the data characters in a buffer, the characters will be lost before the program has time to capture it.

A circular buffer is used to resolve this problem by providing temporary storage for the received data characters until such time that the PC can catch up with stream of input characters. The circular buffer is controlled by three integer variables: *CircIn*, *CircOut*,

and *CharsInBuf*. *CircIn* points to the next character that the interrupt routine places into the input buffer, and *CircOut* points to the next character to be taken out of the buffer. *CharsInBuf* reflects the number of characters waiting in the buffer.

Figure 3-2 shows the operation of the circular buffer. When no characters are in the input buffer, *CircIn* and *CircOut* are equal and *CharsInBuf* is zero. When data arrives at the serial port, the interrupt routine adds the incoming characters to the buffer and increments both *CircIn* and *CharsInBuf*. This buffer is "circular" because when the end of buffer is reached, *CircIn* is reset to 1, or the beginning of the buffer.

The procedure *GetCharInBuf* is used to determine whether *CharsInBuf* is greater than zero, indicating that characters are present in the buffer. If *CharsInBuf* is greater than zero, the characters in the buffer are removed, *CharsInBuf* is decremented, and *CircOut* is incremented. Thus, *CircOut* constantly polls *CircIn* to determine when there are no characters remaining in the input buffer. At present the buffer size is limited to 1 Kb, which should be more than adequate for most communications purposes. However, the buffer size can easily be changed in the TMC source program.

### **3.4 Interpreter**

The interpreter performs line-by-line syntax analysis of the task program. The basic method used is called "top-down parsing"

since it consists of top-down attempts to reconstruct the generating steps from their start symbol to the final symbol. The task program is a sequence of statements, each of which defines an action to be carried out by the task processor. In this sense, the TMC is a sequential programming language, i.e., statements are executed sequentially in time and never simultaneously. In general, the parser is intended to recognize six kinds of task program statements: block statements, conditional statements, repetitive statements, command packet statements, data packet statements, and built-in function statements. The parser is a direct translation of the syntax graph illustrated in Figure 3-3, which reflects flow control during the process of parsing a sentence. The detailed format of the command packet statement and the data packet statement are discussed in Chapter 4.

### 3.5 Task Processor

The task processor is of crucial importance to the TMC program since it analyzes and executes program statements. This processor encompasses three basic functions. The first function calls the *send\_packet* procedure to transmit packets if the statement is either a command or data packet.

The purpose of the second processor function is to manipulate program flow control. When a conditional statement is encountered, the processor evaluates the condition. If the condition is not matched, it sets the skip flag to *FALSE* so that the

subsequent statements are skipped until the reserved word *ELSE* is encountered, following which the skip flag is reset to *TRUE*. On the other hand, if the condition is matched, the converse of the above operation is carried out. The skip flag is set to *TRUE* and is reset to *FALSE* only when the reserved word *ELSE* is encountered. In both cases, the skip flag will reset to *TRUE* when the reserved word *ENDIF*, indicating the end of a conditional statement, is found.

For repetitive statements, a counter is used to track the number of times a following statement, including block statements, must be executed. The counter is decremented each time it executes the statement. This process continues until the counter reaches zero, following which the next statement will be executed in continuation of the execution process. If the statement followed by the repetitive statement is a block statement, or a series of statements enclosed by the reserved words *BEGIN* and *END*, then a pointer is used to indicate the beginning of the block statement so that whenever *END* is encountered, control can again be returned to the first statement.

The last task processor function is to execute the built-in functions, a number of which are provided to enhance the TMC task scheduling performance. These functions include *WAIT*, *DELAY*, *PRINT*, *PRINTF*, *SAVEON* and *SAVEOFF*.

## ***WAIT***

The *WAIT* function is used to suspend the task program execution until a response expected from the microcontroller is

received by the *receive\_packet* procedure. A timer is set to 1 second and if the response is not received within that time, a decision statement like *IF NORSP THEN* or *IF RSP THEN* can be used to determine the next set of tasks to be performed. An error message can also be displayed using the *PRINT* function discussed below.

### ***DELAY***

The *DELAY* function, based on the *DELAY* procedure defined in Turbo Pascal built-in procedure and function library [TURB 88], is used to delay any period of time defined by the user in seconds.

### ***PRINT AND PRINTF***

The *PRINT* function displays messages in a message window, which may be either a warning or a reminder to the operator. On the other hand, *PRINTF* writes the message to a response file as a comment for received data, e.g., the address of the microcontroller where the data came from and its meaning.

### ***SAVEON AND SAVEOFF***

The *SAVEON* and *SAVEOFF* functions are used, respectively, to turn on and off the save response flag. All received responses are saved to a response file when the response flag is on. However, in either case (*PRINT* or *PRINTF*), received responses are displayed in the response window.

### 3.6 Screen Editor

The *editor* procedure provides a built-in screen editor for the editing of the task program. The user can edit the program within the editor window, a window consisting of 19 horizontal lines. When all 19 lines within the window have been filled, the screen automatically scrolls up one line.

The *editor* procedure continuously checks the keyboard to see if a key has been pressed. The keys are divided into three groups: cursor movement keys, function keys, and character keys. The cursor movement keys consist of the four arrow keys, i.e., left, right, up, and down, plus the PgUp and PgDn keys, and are used to control the cursor movement on the screen. The function keys include F7 and F8 as well as the DEL and INS keys. Pressing F7 deletes one line, while pressing F8 causes the insertion of one line. The DEL and INS keys are used, respectively, to insert or delete one character. The last group of keys available for editing includes all of the character keys.

The *WhereX* and *WhereY* functions are used to track the current cursor position. They are updated each time a key is pressed. The *GotoXY* function is used to move the cursor to the new position specified by the coordinates X and Y. The *display\_task* procedure is used to update the editor screen when required.

### 3.7 System Library

The system library provides a support environment to allow novice users to prepare task programs. It consists of a device library and a task library. The device library contains a list of all the microcontrollers attached to the network. Devices (or microcontrollers) can be added to or deleted from the library list. The task library consists of all the tasks that each microcontroller can perform. The task list is updated by the system developer in a method similar to using the editor.

The syntax of each task command packet is checked by the interpreter. In this manner, the novice user can be assured that each task command packet used is presented in the correct syntax. Each listed task also comes with a brief description of the actions it performs. These descriptions can also be used as a task commands by the novice user for task editing. In other words, the detailed (and unfriendly) packet format is hidden from the user. In this fashion, the novice user can edit tasks using friendly "high level" statements.

It is important to note that the system library is only as current as the user makes it. If new controllers are added or deleted, these changes will not be reflected in the library unless the system developer updates the library accordingly. The procedure for updating the task library is similar to the editing procedure. After the task command is entered the cursor moves to the description field instead of moving to the next new line.



The entire library utilizes a dynamic data structure, with memory allocated for the creation of new device task lists only when the added microcontroller is placed on the device list. Similarly, memory is deallocated when a microcontroller is disconnected from the network. Accordingly, memory allocated for the task library is always kept to a minimum. The data structure for the system library is shown in Figure 3-4.

### **3.8 File Handling**

The file handling routines consist of the following file and directory management procedures and functions:

- Load file: loading a file from the TMC list of support files in current directory;
- New file: creating a new file for editing;
- Save file: saving the current editing file;
- Write file: saving the current editing file under another file name without modifying the name currently in use;
- Change Directory: changing the default directory to another directory.
- Exit to DOS: temporarily exiting the TMC program to DOS, returning to TMC by typing "Exit"; and
- Quit: exiting the TMC program, returning control to DOS.

These features are implemented by calling the corresponding standard Pascal library procedures and functions, including ChgDir, Findfirst, FindNext, and GetDos[TURB 88].

### **3.9 Network Monitoring**

The network monitoring procedure is used to poll (send a testing packet to) all the microcontrollers in order of their address. The results provide current microcontroller status (i.e., active or inactive) so that the user can be readily aware of the address of the microcontrollers in use.

### **3.10 User Interface**

One of the principal motivations underlying the design of the TMC is that novice users can use the program with relative ease. A significant portion of the TMC program is devoted to the creation of an effective user-program interface. The result is a user-friendly, menu-driven program which allows anyone to easily follow simple directions to achieve desired actions. The user interface routines consist of procedures and functions for the creation of a main header line menu, pull-down menus, pick-from-list menus, and pop-up windows.

### 3.11 Miscellaneous

All of the procedures and functions which do not fit into the above categories are placed in this section, including the procedures to set up the program environment and to create help and error messages, as well as check key input and sound generating functions.

The procedure *ChgVar* is used to change or set up the program environment, including the communication parameters (baud rate and COM port), and to change the task programming mode from user to command and vice versa. It also turns on and off the save response flag. Changes can be made by simply pressing the Enter key to toggle the setting.

Context sensitive help and error messages are available to the TMC user. Help files are first generated using the *TMC\_help.pas* program. To create the help function, the user assigns a constant equal to the help name and defines a record for that assignment. Constants must be defined consistently in both the *TMC\_help.pas* and TMC programs.

Unlike help messages, error messages are stored in memory. There is a defined array, *ErrorMessStr*, which contains the messages that are popped-up when an error occurs. The main routine checks for a non-zero error number and reports the error when it occurs.

Finally, there are several key decoding routines used to decode the function keys, the Esc key, the Space bar, and the Return (Enter) key. Procedures are also included to generate

sound when an error occurs or to provide a warning to the user in such situations as when an exit program command has been received and files have not been saved.

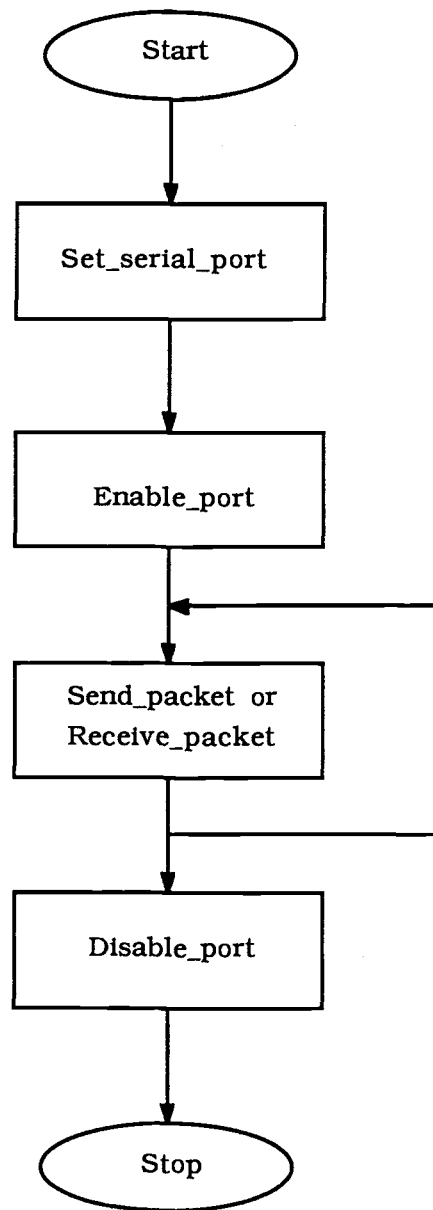
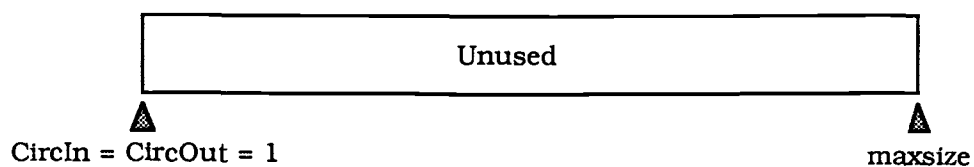
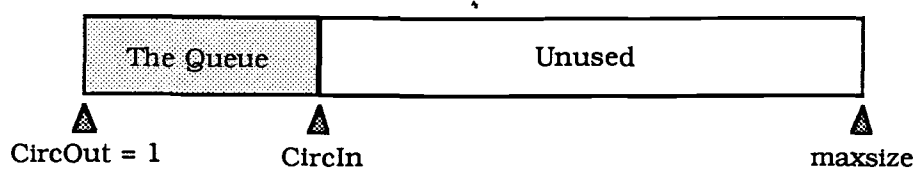


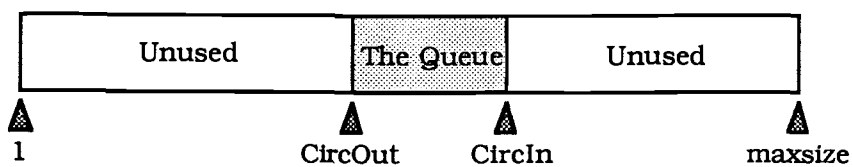
Figure 3-1. Flowchart For Establish Asynchronous Communication



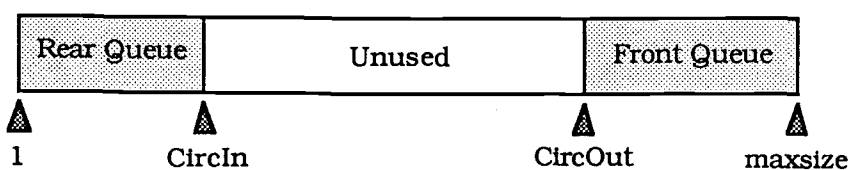
(a) An empty queue



(b) The queue with unread characters stored



(c) The queue in (b) at a later time



(d) The queue in a wrapped state

Figure 3-2. Circular Queue Operations

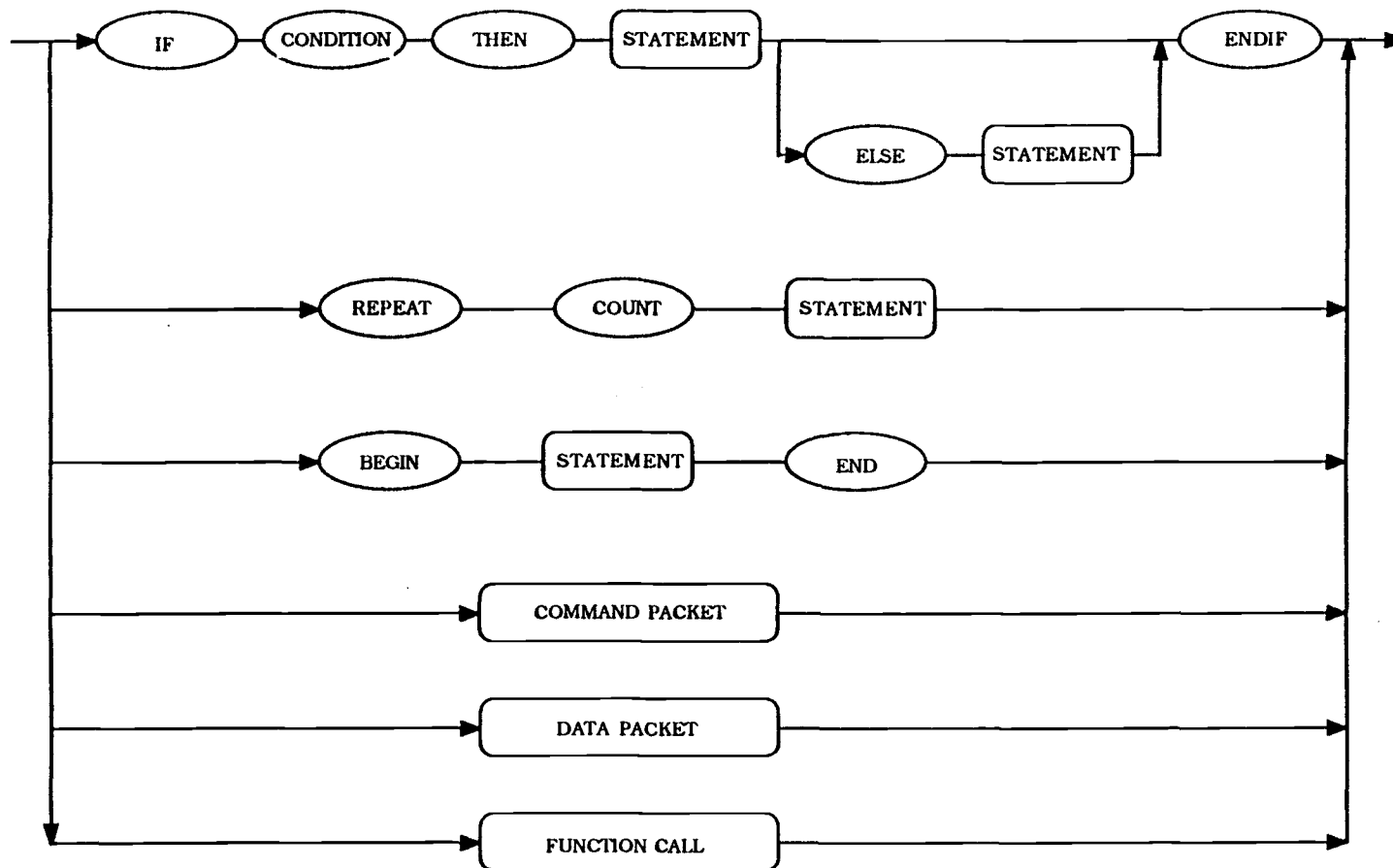


Figure 3-3. Syntax Graph For A TMC Statement

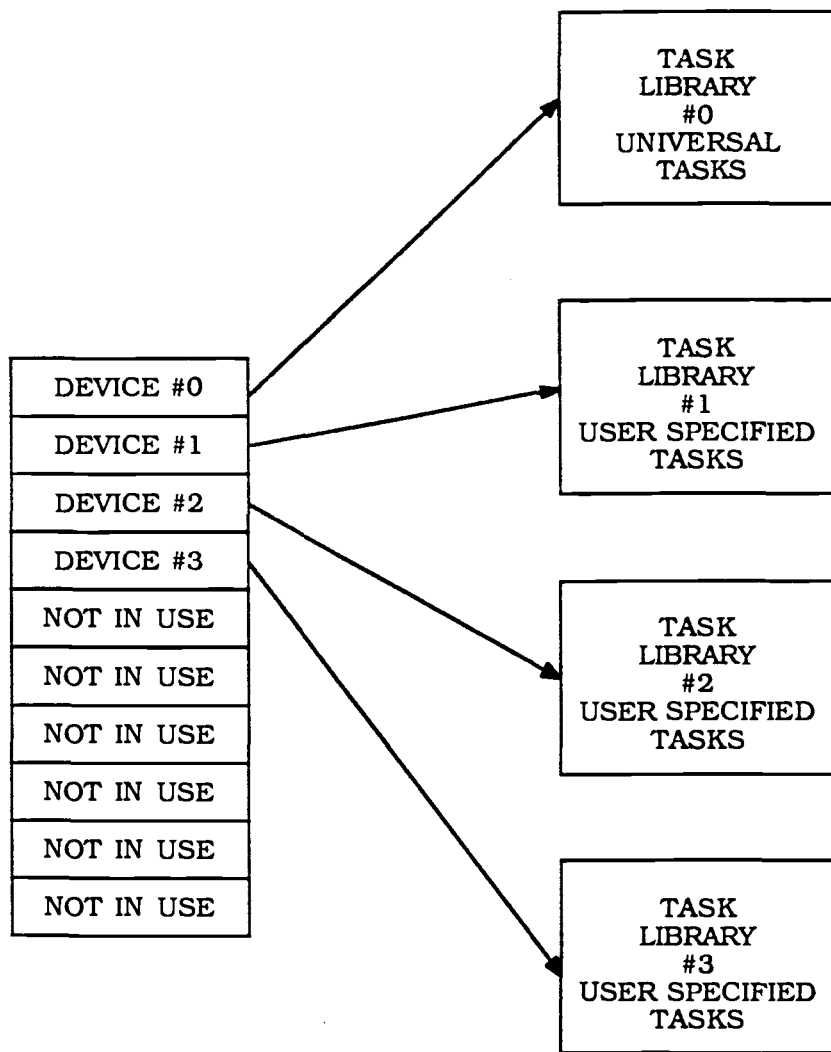


Figure 3-4. Data Structure For TMC Library



## **CHAPTER 4**

### **USER MANUAL FOR TMC SOFTWARE**

#### **4.1 Overview**

The TMC program, designed to run on IBM-PC/XT/AT/386 or compatible computers, provides a number of features to aid the user with task scheduling and system development. Software operations, including software installation, the use of menu commands, and the structure of the task program, are provided in the following detailed descriptions.

#### **4.2 Installation**

The TMC software packet is composed of TMC.EXE, TMC.HLP, and TMC.LIB, which include, respectively, the executable file for the TMC, the help file, and the library file. It is recommended that software for the TMC be installed in its own subdirectory. The following steps will complete this installation :

- (1) insert the TMC diskette in drive A,
- (2) change to the root directory (cd \) on C: drive,
- (3) make a subdirectory for TMC (md TMC),

- (4) change to the \TMC directory (cd TMC), and
- (5) copy all files from the diskette to the \TMC subdirectory (copy a:tmc.\* c:\TMC).

### **4.3 Getting Started**

To start the TMC, enter "TMC" at the DOS prompt, after the introductory message a "plug and play" system menu is displayed. This menu provide three option: Main Menu, Run and Quit.

- (1) Main Menu : The main menu allows the user to go into the integrated development envirnoment.
- (2) Run : The Run option allow the user to run the default task program (TMC.CMD) without going into the integrated development envirnment. This option provides a faster and simpler way for noivce user who just wants to run a program already developed by the system developer.
- (3) Quit : Exit the TMC program.

### **4.4 Using the Menu System**

When main menu option from the system menu is selected, TMC will enter the integrated development environment, shown in Figure 4-1. The main screen consists of four parts: a main menu, the editor window, the response window, and the quick reference

(Quick-Ref) line.

To provide familiarity with the TMC system, some of the navigating basics are as follows:.

- (1) Select a highlighted capital letter to choose a menu item, or use the arrow keys to move the cursor to the option, then press the enter key.
- (2) Press the esc key to exit the menu.
- (3) Use the right and left arrow keys to move from one menu to the next.
- (4) Press the F1 key for context sensitive information relative to the cursor position (i.e., help in menu selection or editing).
- (5) Press the F10 key to invoke the main menu.

#### **4.4.1 Main Menu**

The TMC main menu bar, located at the top of the main screen, offers six choices:

- (1) File: File handling operations (loading, saving, creating, or writing to disk), the manipulation of directories (changing), quitting the program, and invoking DOS.
- (2) Edit: Allows the creation and editing of task program files.
- (3) Execute: Checking and running the task program.
- (4) Setup: Allows changes in communication parameters, sets

the operating environment, and enables or disables the response auto-save function.

- (5) Library: Allows updating of both the device list and the corresponding task library.
- (6) NetInfo: Displays the status of the entire network.

Note that two of the main menu items provide only one option: Edit simply invokes the editor and NetInfo displays only the network status. However, the other menu items lead to pull-down menus with a large number of options.

#### **4.4.2 Quick-Ref Line**

The Quick-Ref line at the bottom of the screen displays "hot keys" which perform specified actions. To select one of these functions, press the listed key:

- (1) <F1> is help. Pressing this key opens a help window which provides information on the TMC commands.
- (2) <Esc> is cancel. Pressing this key cancels the current selection.
- (3) <Arrow>. All of the arrow keys are movement keys, moving the cursor either left, right, up, or down.
- (4) <Enter> is select. Pressing this key selects a command.
- (5) <F10> is the menu key, calling the main menu.

### **4.4.3 Editor Window**

To get into the editor window, select (move the cursor to) the edit option on the main menu and press Enter. Statement are entered and edited line-by-line (active lines are indicated by a highlighted bar) in the editor window. The screen scrolls when the editor window is full. The editor commands used most often are summarized below:

- Scroll the cursor though your text with the up/down, left/right, and pgup/pgdn keys.
- Insert lines with the F7 key.
- Delete lines with the F8 key.
- Insert characters with the Ins key.
- Delete characters with the Del key.

There are two different approaches to task program editing, one for novice users and the other for users experienced in system development. Each is explained in the sections 4.4.3.1 and 4.4.3.2 respectively.

#### **4.4.3.1 User Mode Task Editing**

In the user mode, the response window on the left side of the screen is replaced by a task list window, as shown in Figure 4-2. Task programming is accomplished as follows:

- (1) Use the up and down arrow keys to move the highlighted bar to the line where the desired statement is to be entered.
- (2) Move to the task list window by moving the highlighted bar with the right arrow key and select a predefined task from the task list. The selected task will automatically be transferred to the task program editor at the desired position.

Repeat these two steps until the entire task program is completed. Note that typing is not necessary in these operations. The default task list is a universal task list, listing all of the tasks that can be applied to each of the remote microcontrollers in the network, including the "friendly blink" and "abort" tasks, as well as conditional statements, block statements, repetitional statements and built-in functions. Table 4-1 provides the library of the universal tasks [LORN 89] available for the system developer to enter into the TMC system library when required while in the task list window, pressing the Esc key will pop-up the device list. Selecting a device from the list will replace the current task list with the task list for the selected device. In the user mode, the F2 key is used for program indentation. Each time the F2 key is pressed, the selected line will be indented by adding three spaces at the beginning of the line. A maximum of two level indentation is allowed. Pressing the F2 key again will cancel the indentation.

#### **4.4.3.2 Command Mode Task Editing**

In the command mode, task editing resembles text editing in most conventional line editors. The TMC task editor has a simple structure. The maximum line width is 40 characters. Each line can contain single or multiple task commands or data packets. However, a long packet is not to be split into two lines. Users should break up longer data packets into smaller packets, each with a maximum line length of 40 characters. Press the enter key at the end of each line to proceed to the next line. Each time the enter key is pressed, the syntax of the line is checked by the interpreter, and an error message will be displayed if an error is found. Figure 4-3 shows an example of command mode task editing.

### **4.5 Menu Commands**

The main menu contains the major selections for loading task files, editing, checking syntax, and running the task program. The six options included in the menu are described in the following sections.

#### **4.5.1 File Menu**

The file pull-down menu offers a variety of choices to load

existing files, create new files, and to save files. When a file is loaded, it is automatically placed in the editor. When file editing is completed, it can be saved to any directory under any non-conflicting file name. In addition, this pull-down submenu can be used to change to another directory, temporarily return to the DOS shell, or to exit from the TMC. The file menu is shown in Figure 4-4. File menu functions are described as follows:

- (1) Load: Loads a file when a file name is selected from a list of file choices. In the user mode, the listed files have an ".usr" extension. Similarly, in the command mode the listed files end with a .cmd extension. Users are not able to load a file into the wrong environment. When trying to load another file, the user will be asked to verify if there are already unsaved or modified files in the editor.
- (2) New: Specifies that a new file is to be created. By default, when entering the editor, this file is called either "noname usr" in the user environment or "noname.cmd" in the command environment (this name can be subsequently changed when the file is saved).
- (3) Save: Save the file in the editor to the disk. When the file name is presented, the editor will ask if the file should be renamed.
- (4) Write to: Writes the file to a new name or overwrites an existing file.
- (5) Change dir: Displays the current directory and allows the user to change to a specified drive and directory.



- (6) OS shell: Leaves the TMC and returns to the DOS prompt.  
To return to the TMC, type "exit" at the DOS prompt and press the enter key. This is useful when DOS commands must be run without quitting the TMC.
- (7) Quit: Quits the TMC and returns the user to the DOS prompt.

#### **4.5.2 Edit Command**

The Edit command invokes the built-in screen editor. The main menu can be invoked from the editor by pressing F10. However, the source text display remains in the editor window; the user can return to this display by reselecting the edit command.

#### **4.5.3 Execute Menu**

As shown in Figure 4-5, there are three options in the execute menu: check, run, and step.

- (1) Check (in command mode only): The check command invokes syntax checking for the current task program. This is a redundant function since the TMC provides an interpreter to check syntax line-by-line during the process of editing the task program. However, selection of check allows the user to edit the task program in another word

processor and to load it into the TMC only for execution. In this case, it is strongly recommended that the check be used to determine the accuracy of the task program syntax prior to execution of the task program. It should be noted that if an external word processor is used to edit the task program, it should be limited to line lengths of 40 characters per line in order to fit into the TMC editor window. Files created by external word processor are restricted to contain ASCII keys only, control characters are not allowed.

- (2) Run: The Run option is used to execute a task program. The TMC executes the program currently in the editor. The program will normally run to the end unless a run-time error is encountered. The user can stop the execution of the program at any point by hitting the character "P" (Pause) key.
- (3) Step: The step command allow users to step line-by-line through the task program. The TMC will execute a program one line at a time. Each time the enter key is pressed, TMC executes one more line. With this feature program operation can be tested, line-by-line, from start to finish. In addition, the F9 key can be used to execute the same line repeatedly. In this case, the user can transmit the same packet or packets repeatedly without advancing to the next line.

#### 4.5.4 Setup Menu

The Setup menu is shown in Figure 4-6, it contains the settings which determine the operation of the integrated environment, including the communication parameters and the user's environment. All settings are changed by toggling the selection.

- (1) ComPort: This command sets the serial communication port at either COM1 or COM2. The default setting is COM1.
- (2) BaudRate: Data transmission speed is set by selecting the appropriate baud rate at either 300, 600, 1200, 2400, 4800, or 9600 baud. The default baud rate is 300.
- (3) Environment: The TMC provides two different operating environments, user mode and command mode, designed for novice users and expert programmers respectively. In the user mode the actual command and data packet formats are hidden from the user and a greater proportion of high level statements (e.g., "Turn On Switch #1" or "Increase Light Intensity") are used. High level commands are defined and stored in the task library by the system developer. The default environment is user mode.
- (4) Save Response: The save response feature provides an option to record all responses received in a text file. This allows the user to save responses from remote

microcontrollers for subsequent analysis. The responses are saved in a text file with the same name as the task program file, but with a file extension of ".rsp". The save response default is "no."

#### **4.5.5 Library Menu**

The library menu allows the system developer to maintain a library, including devices and their corresponding tasks, by updating the library when new microcontroller units are added to the system. Existing devices and tasks can also be renamed or deleted and new commands can be inserted into the task list. When updating the task list, two types of information are required: the command packet and a description of its functions. The latter also serves as a high level command for the novice user in the user mode. Similar to editing a task program in the command mode, when a packet is entered, the interpreter checks the packet syntax to assure that its format is free of error. In this manner, novice users can be assured that the packets associated with the high level commands are also error free. Note that the first device 00, is reserved for universal commands which cannot be replaced or deleted. Though not recommended, associated tasks can be changed or deleted, including words reserved for conditional statements, repetitive statements, block statements and such universal tasks as friendly blink and abort task. The device library and task library are shown in Figure 4-7 and 4-8 respectively.

#### **4.5.6 NetInfo Command**

The NetInfo command provides status information on the entire network, displaying the address, status (active or off), and name of each device. The NetInfo. window is shown in Figure 4-9.

### **4.6 Basic Task Program Statements**

The TMC consists of six types of statements: command packet statements, data packet statements, repetitive statements, conditional statements, block statements, and built-in function statements.

#### **4.6.1 Command Packet Statements**

The command packet is also known as TASK. The general format for command packet statements is as follows:

```
{ AA P TT S DD DD DD DD DD /}
```

The curly braces are characters which indicate the beginning and the end of a packet. The other parameters in this format are described in the following sections:

#### 4.6.1.1 Device Address

The device address, AA, is a pair of hexadecimal numbers which identify the destination of the command packet. For example, {07:15.} addresses a remote device controller unit with an address of 07. Note that address 00 is reserved as a universal address, addressing all of the remote microcontrollers in the network. Therefore, {00:15} is directed to all remote microcontrollers.

#### 4.6.1.2 Prefix

The prefix, P, is a single pre-execution control character used to specify the execution priority of a task. Three types are available as follows:.

- ":" Queued Task. The task is placed on the task queue. It is executed only when it reaches the head of the queue and the execution of the current task is completed.
- "?" Synchronized Task. This task, similar to the queued task, is placed at the end of the task queue. It is executed only when the task reaches the head of the queue and the execution of the current task is completed. However, execution does not begin until a synchronizing command from the TMC is received.
- "!" Immediate Task. This is the highest priority task.

These tasks are not placed in the queue, but are run immediately. The current running queue task (if any) is temporarily suspended and resumed after completion of the immediate task. An immediate task cannot be interrupted by another immediate task. If an immediate task is issued while another immediate task is running, the second immediate task will be ignored. The execution of an immediate task can only be stopped before completion by issuing an abort task command.

For example, with a parameter of {09:15.}, Task 15 is placed on the task queue of microcontroller unit 09 and will be run upon completion of all other queue tasks already stored in the task queue. The parameter {07?13.} indicates that task 13 is placed on the task queue of microcontroller unit 07 and will be run upon completion of all other queue tasks already stored in the task queue and the receipt of a synchronized signal. In turn, {05!0A.} indicates that task 0A is to be executed immediately by microcontroller unit 05. The currently running queue task is suspended.

#### **4.6.1.3 Task Number**

The task number, NN, is a two-character hexadecimal task number which determines which task is to be performed. The queue-management task and other standard utilities are usually provided by the microcontroller operating system. Additional

custom and specific application tasks are defined or written by the system developer.

#### 4.6.1.4 Suffix

Three suffixes, S, designations determine the terminating mechanism:

- "." indicates that the task is discarded after execution.
- "+" indicates that the task will be requeued following execution and run again when it reaches the head of the queue. Requeued tasks always remain in the queue until they are discarded by the host.
- "\*" indicates that the task will be queued a certain number of times. The number of times it will be repeated is given by the first argument (i.e., 01 = once, 00 = 256 times). After the specified number of repetitions, the task will be discarded by the local operating system.

For examples, {07:15.} indicates that task 15 will be placed on the queue and will be executed only once. Following execution, the local operating system will discard task 15. A designation of {07:15+} indicates that task 15 will be requeued following execution. It will be executed and repeated until it is discarded by the system scheduler. In the designation {07:15\*03}, task 15 will be requeued twice and executed three times. Following the third



execution, task 15 is discarded by the local operating system.

#### **4.6.1.5 Data Field**

The data field, DD, is optional and consists of 0 to 5 pairs of hexadecimal numbers. These numbers indicate that the host computer is passing data, arguments, or encoded instructions to a remote microcontroller for use in executing a task. Some of the tasks do not require arguments and the argument is therefore omitted. If the task is run in the "\*" mode, only four arguments can be specified because the first argument is used to determine the number of times the task is to be run. Examples of valid commands include: {07?13+1234}, {07!13.1122334455}, {08!15.} and {07?04\*0311223344}.

#### **4.6.1.6 Echo Requests**

The echo statement, "/", is also optional. Any command which includes the "/" character will be echoed back (without the "/"}) to the host computer from the microcontroller. Brackets ("[ ]") rather than curly braces ("{ }") are used to enclose the command. This echo allows the host to verify the correct reception of the command.

#### **4.6.2 Data Packet Statements**

The command packet statement can include five data pairs of hexadecimal numbers. However, in cases when it is necessary to send longer data streams to a remote microcontroller, then the data packet statement should be used. The format of data packet statement is as follows: [DATA].

Anything enclosed within paired brackets is a data packet statement. The data can be a string of characters of a length of 40 characters, or the length of the editor window. Data with a length longer than 40 characters is to be split into two or more packets. For example:

[012333456789ABCDEF]

[THIS IS A TEST]

#### **4.6.3 Conditional Statements**

The conditional statement specifies that a statement can be executed only if a certain condition is true. If it is false, then either no statement is executed or the statement following the reserved word "ELSE" is executed. Note that TMC provides two built-in conditions, RSP (responded) and NORSP (not responded), and an operator to compare greater than, less than, or equal to conditions. For example:

```
WAIT
IF NORSP THEN
    {03:05.}
ELSE
    {03:06.}
ENDIF
WAIT
IF DATA > 10 THEN
    {04:0A.}
ELSE
    {04:0B.}
ENDIF
```

#### **4.6.4 Repetitive Statements**

Repetitive statements specify that certain statements are to be executed repeatedly. The argument for a repetitive statement specifies the number of repetitions required. For example:

```
REPEAT 3
    {0A:09.12345678}
```

#### **4.6.5 Block Statements**

A block statement is used if more than one statement is

specified. It consists of any number of statements enclosed within the reserved words, BEGIN and END, specifying that the component statements are to be executed in the sequence in which they are written. For example:

```
REPEAT 5
  BEGIN
    {01:03.} {08:0A.12345678}
    {04:0B./}
    {05:0C.}
  END
```

#### **4.6.6 Built-In Function Statements**

The following are the TMC standard functions:

- (1) WAIT: Wait for a remote microcontroller to respond. A typical expected response is the acknowledge message from a microcontroller to indicate that some task has been completed successfully. If the remote microcontroller does not respond within one second following the issuance of the wait command, then execution is suspended and a "NO RESPONSE" error message is displayed. The process can continue by pressing the enter key.
- (2) DELAY: Delay for certain amount of time. The delay time is specified by the user as an argument. For example,

DELAY 2.5 will cause a delay of 2.5 sec. This function is generally used to wait for the completion of a task prior to issuing an order for another task. For example, a task to move a robot arm from position x to position y could take five seconds to accomplish and the next task will not be issued until the robot arm reaches position y. Inserting a DELAY 5 statement between the two tasks will accomplish this goal.

- (3) PRINT: The print statement allows the user to print a message on the screen. Generally, this message is to inform the operator that a manual control, e.g., push a button or turn on a switch, must be carried out. It also serves as a warning message to operators, including statements such as "temperature is too high."
- (4) PRINTF: The printf statement allows the user to add comments to a response file. Recall that this file consists of responses received from remote microcontrollers. The user can add such comments as "these are the responses from unit 0A" or "this is the temperature reading." By adding comments, data recorded in the response file can be analyzed easily at a later time.
- (5) SAVEON: The saveon statement saves the response, informing the TMC processor that subsequent responses from remote microcontrollers must be saved in the response file.
- (6) SAVEOFF: Disables saveon, causing the TMC to ignore subsequent responses.

Task #	Task Name	Format	Function
'00'	RSET	{00:00.} or {%}	resets taskmaster
'01'	CLRQ	{00:01.}	clears queue
'02'	ABRT	{00!02.}	aborts task during execution
'03'	SYNC	{00!03.}	starts a task waiting for sync-signal
'04'	HALT	{00!04.}	halts queue processing
'05'	CONT	{00!05.}	resumes halted queue processing
'06'	KILL	{00!06.}	kills task during execution
'07'	NULL	{00:07.}	no operation
'08'	INFT	{00:08.}	never ending task
'09'	DUMQ	{00:09.}	displays contents of task queue
'0A'	POLL	{00:0A.}	displays address and event register
'0B'	HTOM	{00:0B.23D0}	transfers data host -> internal memory
'0C'	MTOH	{00:0C.21}	sends contents of int. byte to host
'0D'	HTOX	{00:0D.F00380}	transfers data host -> ext. memory
'0E'	XTOH	{00:0E.F002}	transfers data ext. memory -> host
'0F'	PAUS	{00!0F.}	suspends task during execution
'10'	RESM	{00!10.}	resumes a paused task
'11'	DELT	{00!11.15}	delete task by task number
'12'	DELE	{00!12.1B}	delete task by entry number
'13'	DLAY	{00:13.05}	delays queue processing
'14'	BEEP	{00:14.}	sends an audio signal to the host
'15'	BLNK	{00:15.}	visual signal: friendly blink
'16'	DUMP	{00:16.}	downloads INTEL Hex File
'17'	P1TH	{00:17.}	send contents of P1 to host
'18'	HTP1	{00:18.FF}	writes byte to internal port 1
'19'	SYNT	{00:19.15}	synchronizes queue task by task number
'1A'	ADVC	{00!1A.}	advance sync task and change status
'1B'	GOTO	{00:1B.3B7C}	jumps to address given
'1C'	EXIT	{00:1C.52F3}	exits from XMSIBY to address given
'1D'	BAUD	{00!1D.40}	sets the baud rate
'20'	TIME	{07:20.235959}	sets the system clock
'21'	TINT	{07:21:000003}	sets a relative timeout value
'22'	TOUT	{07:22.150000}	sets an absolute timeout value
'23'	TCNT	{07:23.D0}	sets timer status register (23H)
'24'	SNTH	{07:24.}	displays all timer registers
'26'	TTOH	{07:26.}	sends the system time to the host

Table 4-1. List of Universal Tasks

File	Edit	eXecute	Setup	Library	NetInfo
Task Program			Echo & Response		
<F1>- Help   <Esc>- Cancel   <Enter>- Select   <Arrow>- Move   <F10>- Menu					

Figure 4-1. The Main Menu

File	Edit	eXecute	Setup	Library	NetInfo
Task Program			Task List		
<pre> SAVE RESPONSE PRINTF "THIS IS THE DATA FROM UNIT #1" REQUEST FOR DATA WAIT FOR RESPONSE IF DATA &gt; 100 THEN   PRINT "WARNING : TEMP. TOO HIGH" ENDIF IF DATA &gt; 80 THEN   TURN THE HEATER OFF ENDIF IF DATA &lt; 70 THEN   TURN THE HEATER ON ENDIF           </pre>			<pre> REQUEST FOR DATA TURN THE HEATER ON TURN THE HEATER OFF           </pre>		
<F1>- Help   <Esc>- Cancel   <Enter>- Select   <Arrow>- Move   <F10>- Menu					

Figure 4-2. User Mode Task Editing

File	Edit	eXecute	Setup	Library	NetInfo
Task Program			Echo & Response		
<pre> SAVEON PRINTF "THIS IS THE DATA FROM UNIT #1" {01 : 01.} WAIT IF DATA &gt; 100 THEN   PRINTF "WARNING : TEMP. TOO HIGH" ENDIF IF DATA &gt; 80 THEN   {01 : 02.} ENDIF IF DATA &lt; 70 THEN   {01 : 03.} ENDIF           </pre>					
<F1>- Help   <Esc>- Cancel   <Enter>- Select   <Arrow>- Move   <F10>- Menu					

Figure 4-3. Command Mode Task Editing

File	Edit	eXecute	Setup	Library	NetInfo
<div> <div>Load</div> <div>New</div> <div>Save</div> <div>Write to</div> <div>Change dir</div> <div>OS shell</div> <div>Quit</div> </div>	Task Program		Echo & Response		
<F1>- Help   <Esc>- Cancel   <Enter>- Select   <Arrow>- Move   <F10>- Menu					

Figure 4-4. The File Menu



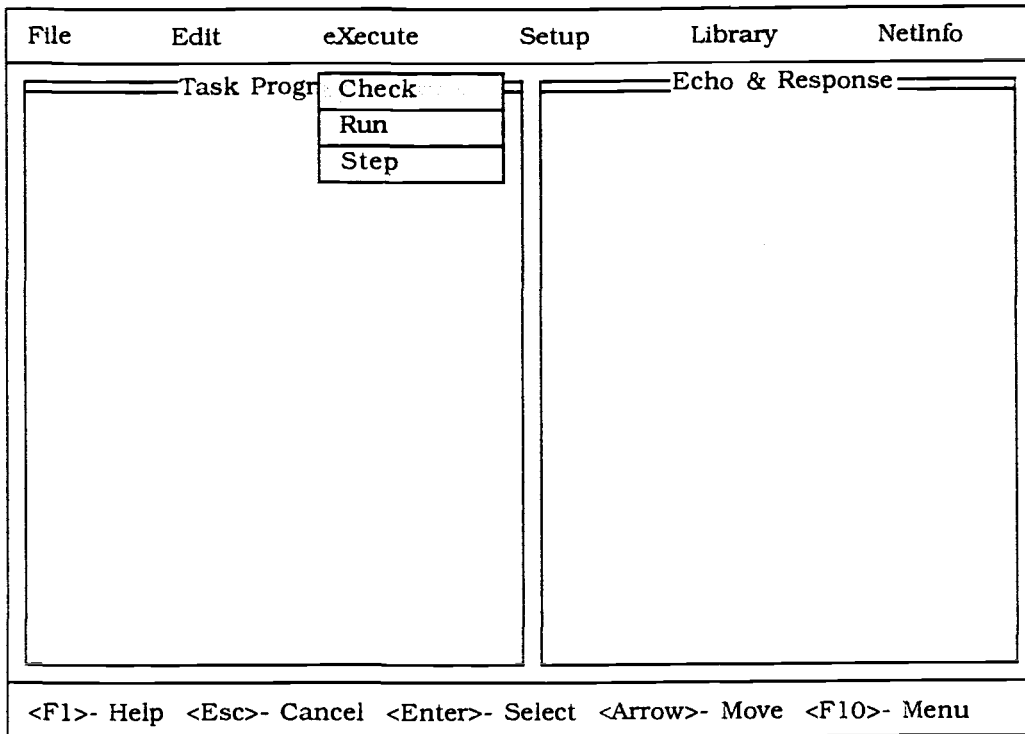


Figure 4-5. The Execute Menu

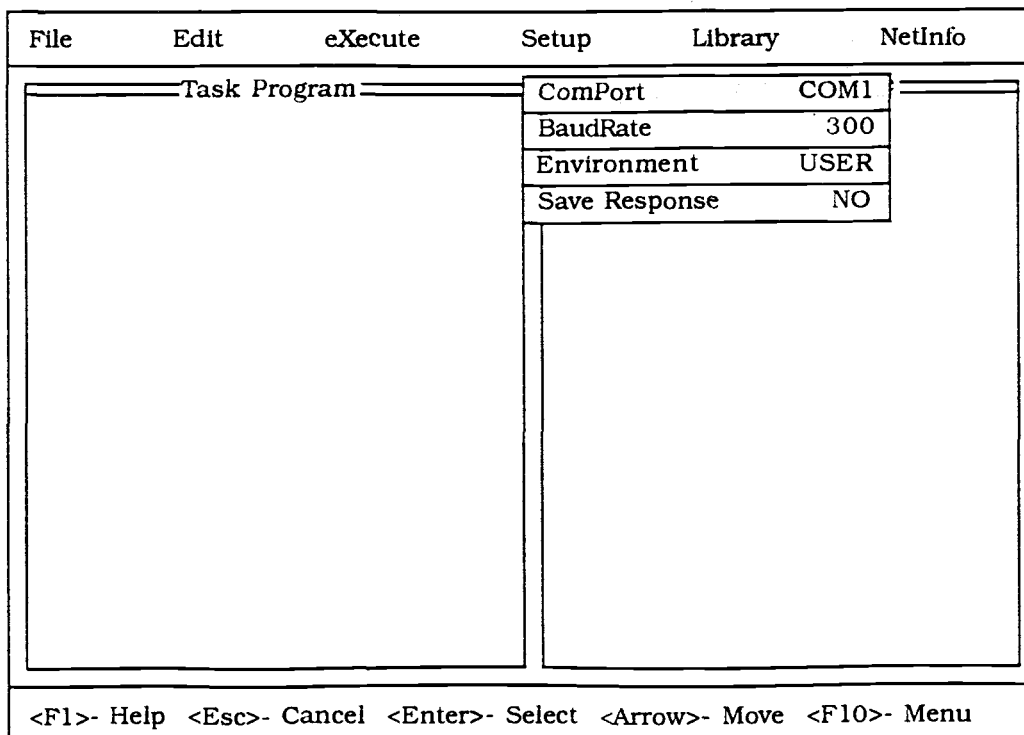


Figure 4-6. The Setup Menu

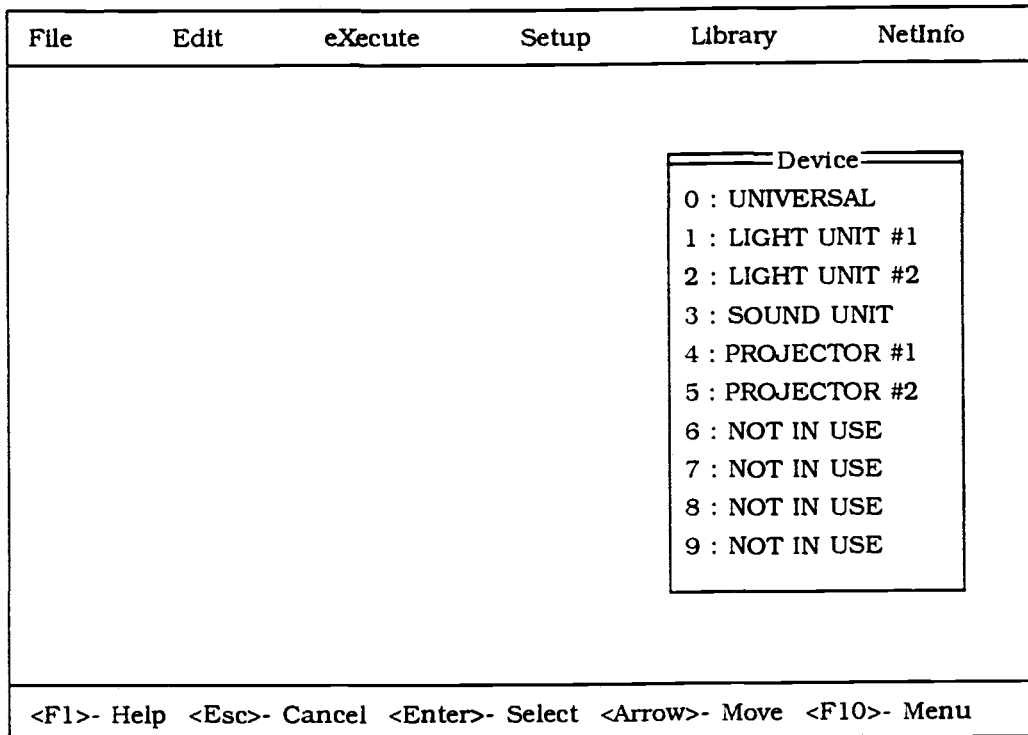


Figure 4-7. Device Library

File	Edit	eXecute	Setup	Library	NetInfo
COMMAND		ACTION			
{01 : 01.}		TURN ON PROJECTOR #1			
{01 : 02.}		TURN OFF PROJECTOR #1			
{01 : 03. 31370000}		PHASE IN PROJECTOR #1			
{01 : 03. 37310000}		PHASE OUT PROJECTOR #1			

<F1>- Help   <Esc>- Cancel   <Enter>- Select   <Arrow>- Move   <F10>- Menu

Figure 4-8. Task Library

File	Edit	eXecute	Setup	Library	NetInfo																																	
Task Program			Echo & Response																																			
<table border="1"><thead><tr><th>NUM</th><th>STATUS</th><th>DEVICE NAME</th></tr></thead><tbody><tr><td>0</td><td>ACTIVE</td><td>UNIVERSAL</td></tr><tr><td>1</td><td>ACTIVE</td><td>LIGHT UNIT #1</td></tr><tr><td>2</td><td>ACTIVE</td><td>LIGHT UNIT #2</td></tr><tr><td>3</td><td>ACTIVE</td><td>SOUND UNIT</td></tr><tr><td>4</td><td>ACTIVE</td><td>PROJECTOR #1</td></tr><tr><td>5</td><td>ACTIVE</td><td>PROJECTOR #2</td></tr><tr><td>6</td><td>OFF</td><td>NOT IN USE</td></tr><tr><td>7</td><td>OFF</td><td>NOT IN USE</td></tr><tr><td>8</td><td>OFF</td><td>NOT IN USE</td></tr><tr><td>9</td><td>OFF</td><td>NOT IN USE</td></tr></tbody></table>						NUM	STATUS	DEVICE NAME	0	ACTIVE	UNIVERSAL	1	ACTIVE	LIGHT UNIT #1	2	ACTIVE	LIGHT UNIT #2	3	ACTIVE	SOUND UNIT	4	ACTIVE	PROJECTOR #1	5	ACTIVE	PROJECTOR #2	6	OFF	NOT IN USE	7	OFF	NOT IN USE	8	OFF	NOT IN USE	9	OFF	NOT IN USE
NUM	STATUS	DEVICE NAME																																				
0	ACTIVE	UNIVERSAL																																				
1	ACTIVE	LIGHT UNIT #1																																				
2	ACTIVE	LIGHT UNIT #2																																				
3	ACTIVE	SOUND UNIT																																				
4	ACTIVE	PROJECTOR #1																																				
5	ACTIVE	PROJECTOR #2																																				
6	OFF	NOT IN USE																																				
7	OFF	NOT IN USE																																				
8	OFF	NOT IN USE																																				
9	OFF	NOT IN USE																																				
<F1>- Help <Esc>- Cancel <Enter>- Select <Arrow>- Move <F10>- Menu																																						

Figure 4-9. Network Status Window

## **CHAPTER 5**

### **SUMMARY AND RECOMMENDATIONS**

#### **5.1 Summary**

In this study the design and implementation of the Task Master Controller (TMC), a programmable system scheduler for COLAN, a distributed control system, has been performed. The primary goal of this project was to design a powerful, user-friendly scheduler that would allow both the novice user and the system developer to schedule tasks on a host computer. An improved network interface unit (NIU) was also developed to provide a data communication interface between the host computer and the network RS-485 communication bus.

The TMC designed and implemented for this study has the following general characteristics:

- The TMC includes a reliable communication driver to assure communication sessions between the host computer and designated microcontrollers in the network.
- The TMC allows users to edit the task program. For novice users, high level language statements are used in place of the low level symbolic statements used by experienced system developers.

- The TMC provides an interpreter which checks the syntax of task program statements.
- A task processor is used to execute the task program.
- Conditional and repetitive statements are provided for program flow control.
- Helpful built-in functions, such as "wait for response" or "print message", have been included to enhance system performance.
- The TMC has file handling commands to enable the user to manage the host computer system.
- The TMC provides a friendly menu-driven style of user interface. Users can activate any operation by simply pressing a key.

## **5.2 Recommendations for Future Research**

The TMC is a preliminary design of a programmable system scheduler for COLAN. Areas of concern which require further design and improvement are listed below:

- Flow control: At present, the TMC provides only two basic flow control statement, i.e., conditional and repetitive statements. Additional flow control statements which should be added include a For/Next statement and a While statement. Moreover, the TMC does not currently support nested flow control statements, which is a restriction

which should be lifted in future developments.

- **Variables:** The TMC does not currently include variables and it would be desirable to have the ability to declare a number of different variable types, such as integer, real, or character. The data received from remote microcontrollers could then be assigned to these variables. At present it is assumed that the data received is expressed in integers and is assigned to a default integer variable, DATA.
- **Arithmetic and boolean operations:** Arithmetic and boolean operations can be added to the TMC to perform these types of operations on the received data. Decision could then be made, based on the results of these operations.
- **Subroutines and functions:** Subroutines and functions are two structures that should be added to the TMC. The current program version allows only the sequential execution of the user's program. Repetitive statements must be executed in a loop or be rewritten for each execution. The inclusion of subroutines would provide an easier means of accomplishing repetitive commands, and would make the program code more readable.
- **Comments:** Comments should be allowed in the user program, especially since the program is written in a low level language format. Comments would help other users understand the program and make it more readable.

The implementation of the above recommendations would require an evolutionary modifications of the interpreter and the task processor.

## BIBLIOGRAPHY

- [BORL 88] Borland International, *Turbo Pascal Reference and User's Guide V 5.0*, Scotts Vally, CA, Borland Int., 1988.
- [COTT 80] I.W. Cotton, "Technologies for Local Area Computer Networks," *Computer Networks*, Volume 4, November 1980.
- [EUM 87] D. Eum, *COLAN III, A Control-Oriented LAN Using CSMA/CD Protocol*, unpublished master's thesis, OSU, Corvallis, OR, 1987.
- [FATH 83] E.T. Fathi and Moshekrieger, "Multiple Microprocessor Systems: What, Why, and When," *IEEE Computer*, March 1983.
- [HERZ 87] J.H. Herzog and T.G. Zhang, "A Design Methodology for Distributed Microprocessors in Real-Time Control Applications," *Preceedings of Second International Conference on Computers and Applications*, Beijing, China, June 1987.
- [KAO 87] S. Kao, *Design of COLAN II, A Control Oriented Local Area Network*, unpublished master's thesis, OSU, Corvallis, OR, 1987.
- [KUMA 89] R.C. Kumar, *COLAN V, A High Performance Local Network for Control and Communication Utilizing a Cmmunication Coprocessor*, unpublished master's thesis, OSU, Corvallis, OR, 1989.
- [LORN 89] E. Lorenz, *TASKMASTER OPERATING SYSTEM Reference Manual*, unpublished project report, OSU, Corvallis, OR, 1988.



- [NORT 85] P. Norton, *The Peter Norton Programmer's Guide to the IBM PC*, Washington, Microsoft Press, 1985.
- [O'BRI 89] S.K. O'Brien, *TURBO PASCAL 5.5, The Complete Reference*, Borland-Osborne/McGraw-Hill, 1989.
- [SLAT 87] M. Slater, *Microprocessor-Based Design, A Comprehensive Guide To Effective Hardware Design*, Mayfield Publishing Company, 1987.
- [THYE 88] Y. Thye, *COLAN IV, A Network for Communication and Control*, unpublished master's thesis, OSU, Corvallis, OR, 1988.
- [WIRT 76] N. Wirth, *Algorithms + Data Structures = Programs*, Prentice-Hall, Inc., 1976.
- [ZHEN 86] Y. Zheng, *A simple Local Area Network, COLAN (Control Oriented Local Area Network)*, unpublished master's thesis, OSU, Corvallis, OR, 1986.

## APPENDIX

## **APPENDIX A**

### **TMC APPLICATION EXAMPLES**

#### **A.1 Application Overview**

Using the various TMC task formats and proper sequencing of tasks allows a great deal of flexibility to accomplish control requirements. A typical application environment in which a distributed control system can be effectively used consists of multiple stepper motors. Each motor is under control of a single microcontroller. The coordinated activity of several motors could be used to position a robot mechanism or other complex mechanical system.

However, the application of COLAN is not limited to a factory environment. COLAN can be used at any place where a process must be monitored and adjusted. Applications such as temperature and humidity control, fire protective sprinkler system, experimental data collection are valid candidates.

The following section provides two simple application examples to demonstrate how to use TMC for Task scheduling.

## A.2 Example 1 : A Museum Project

A museum project is currently being designed and developed at Oregon State University. The museum project provide enhanced controllers for, a cassette player (device #01) multiple light dimmers (device #02) and multiple slide projectors (device #03) to put on an educational performance for the museum visitors. The following shows a portion of the task program used in this project.

```

:
:
:
{01 : 01.}          PLAY CASSETTE MESSAGE
DELAY 20             DELAY 20 SECOND
WAIT                WAIT FOR RESPONSE
IF RSP THEN          IF RSPONSE THEN
    {02 : 01. 31370000}    INCREASE LIGHT#1 INTENSITY
ELSE                  ELSE
    PRINT "NO RESPONSE ERROR"
ENDIF                ENDIF
DELAY 30             DELAY 30 SECONDS
{01 : 01.}          PLAY CASSETTE MESSAGE
DELAY 60             DELAY 60 SECONDS
WAIT                WAIT FOR RESPONSE
IF RSP THEN          IF RESPONSE THEN
    {02 : 02. 37310000}    DECREASE LIGHT#1 INTENSITY
ELSE                  ELSE
    PRINT "NO RESPONSE ERROR"

```

ENDIF	ENDIF
DELAY 30	DELAY 30 SECONDS
REPEAT 3	REPEAT 3 TIMES
BEGIN	BEGIN
{03 : 01. 010000}	PHASE IN PROJECTOR #1
DELAY 10	DELAY 10 SECONDS
{01 : 01.}	PLAY CASSETTE MESSAGE
DELAY 30	DELAY 30 SECONDS
WAIT	WAIT FOR RESPONSE
IF RSP THEN	IF RSPONSE THEN
{03 : 02. 000100}	PHASE OUT PROJECTOR #1
{03 : 03. 010000}	PHASE IN PROJECTOR #2
DELAY 10	DELAY 10 SECONDS
{01 : 01.}	PLAY CASSETTE MESSAGE
DELAY 20	DELAY 20 SECONDS
{03 : 04. 000100}	PHASE OUT PROJECTOR #2
ELSE	ELSE
PRINT "NO RESPONSE ERROR"	
ENDIF	ENDIF
END	END
{02 : 03. 00000000}	TURN OFF LIGHT #1
.	
.	
.	

The program begins with playing the sound of the displayed animal, for example a polar bear, and then waits for 20 seconds to allow for the completion of playing one segment of message on the

cassette tape. When the response (indicated by the end of a segment) from the cassette recorder unit is received, the program turns on a light (light #1), which is the spot light that focuses on the polar bear. The intensity of the light will increased from off to full brightness. Then it is followed by an introductory message from the cassette player. After the message is played, the light intensity is reduced to one-half in preparation for a slide show. At this point the program will enter a repeat loop. Within the loop two projectors will display the slide photograph of the polar bear lifestyle alternatively. The display is accompanied by an explanatory message recorded on the tape. After looping three times (six slides), the polar bear display is complete. The program turns off the spot light that focused on the bear and proceeds on to the next exhibit. Note that the left hand side of the program is the format used by system developer program, the right hand side program is an example of novice user's program format.

### **A.3 Example 2 : A Greenhouse Temperature Control Project**

This program is designed to monitor and control the heating in a series of greenhouses. Each microcontroller unit, for example device #01, is assigned to a greenhouse, and is assumed to be attached to a thermostat that provides it with the temperature. The task is currently not implemented, but it will suffice for an example.

```

SAVEON                                SAVE RESPONSE
PRINTF " THIS IS THE DATA FROM UNIT #1"
{01 : 01.}                            REQUEST FOR DATA
WAIT                                  WAIT FOR RESPONSE
IF DATA > 100 THEN
    PRINT " WARNING : TEMPERATURE IS TOO HIGH "
ENDIF
IF DATA > 80 THEN
    {01 : 02.}                        TURN OFF THE HEATER
ENDIF
IF DATA < 70 THEN
    {01 : 03.}                        TURN ON THE HEATER
ENDIF
PRINTF " THIS IS THE DATA FROM UNIT #2 "
.
.
.

```

This program shows how the conditional statement can be used. The program started with "turn on the save response flag" so that the subsequence responses will be saved. The printf statement places headings on the responses received. The next task is to request for temperature data from greenhouse #1. When the data is received it will check to see whether it is too high (above 100). If it is too high, a warning message will displayed on the screen. The program then determine whether the heater needs to be turned on or off; If the temperature is above 80° C the heater will turned off,

if the temperature falls below 70° C then the heater will be turned on again. The program then proceeds to check the next greenhouse (#2) temperature controller and the same process repeats.