AN ABSTRACT OF THE THESIS OF

Mohammed Rafi for the degree of Master of Science in Electrical and Computer Engineering presented on March 3, 1994.

Title: Implementation of a High Performance Network Node for a Control Oriented Local Area Network

Redacted for Privacy

Abstract approved: _____ James H. Herzog

TASKMASTER is an experimental microcontroller node of a real-time control oriented network which was proposed by James H. Herzog and Tinggui Zhang to demonstrate the feasibility of a task oriented control structure in performing distributed control actions. This study is a continuation of research involving the TASKMASTER network.

A high performance microcontroller - the 32-bit Motorola MC68332 has been used in this study to implement a node of the TASKMASTER network. Use of the MC68332 with its powerful peripheral subsystems offers significant scope for improvement of the overall performance of the network in addition to strengthening its control processing capabilities. An 8-bit microcontroller - the Intel 8052 has also been used to implement a node of the network.

A high-level language C has been used for coding of the operating system of the network which previously has been coded in assembly. In addition to being more readable, use of a high-level language offers other significant advantages such as portability, smaller code development time and code debugging time and the ability to compare different microcontrollers on a common basis. A performance analysis and comparison between the two microcontrollers used and the language used in coding them was also performed using performance measures designed as part of this study.

Implementation of a High Performance Network Node for a Control Oriented Local Area Network

by

Mohammed Rafi

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Completed March 3, 1994

Commencement June 1994

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 March 3, 1994

Typed by Mohammed Rafi

ACKNOWLEDGMENTS

I would like to express my sincere appreciation and gratitude to Prof. James Herzog for his help, guidance and patience during my graduate studies at Oregon State University. I am grateful to Prof. Bella Bose for his advice and help. I would like to thank Prof. Shih-Lien Lu and Prof. Terrence Beaumariage for taking time out of their busy schedules to be present at my thesis defense.

I thank the faculty and office staff at the Electrical and Computer Science Departments, especially Ms. Rita Wells who does a wonderful job of keeping everything running smoothly.

I would like to thank the staff at Canara Bank who were responsible for granting me an educational loan to help pursue graduate studies.

I am thankful to Arjun, Clay, Aninha, Anju, Varsha, Teresa and Richard Amos for their friendship and support.

Finally, I will always be grateful to everybody in my family for their support during my graduate studies and for many other things too numerous to mention here.

TABLE OF CONTENTS

LIST OF FIGURES

IMPLEMENTATION OF A HIGH PERFORMANCE NETWORK NODE FOR A CONTROL ORIENTED LOCAL AREA NETWORK

CHAPTER I
INTRODUCTION

1.1 Local Area Networks

1.1.1 Definition

A Local Area Network is a data communication system which allows a number of independent devices to communicate with each other [TANE 88]. A Local Area Network (LAN) is different from other types of data networks in that the communication is usually confined to a small sized geographic area such as a single office building, an industrial complex or a campus. A classification based on distance has been used in [TANE 88] to describe the boundaries as follows:

- 0-10m Computer Peripherals.
- 10m-10km Local Area Networks.
- 10km+ Wide Area Networks.

Local Area Networks have a few other distinctive characteristics among which are high data rates (0.1 to 100 Mbps) and low error rates (10^{-8} to 10^{-11}). They usually are owned by a single organization.

LANs have figured prominently in research and development for more than 10 years and commercial exploitation has been increasingly active. They offer ways to

provide relatively inexpensive communication between workstations and devices, distributed processing, rapid access to distributed sources of information and sharing of expensive devices and resources.

1.1.2 Network Control

A node is a physical device that may be attached to a shared medium local area network for the purpose of sharing network resources and transmitting and receiving information on that medium. A network control mechanism needs to be devised to take care of allocating access to the shared resources between the multiple nodes. The control mechanism can be centralized or distributed.

In a **centralized control** system, all network activities are supervised by a central controller. This approach has the advantage of reducing processing at each of the other nodes, but network operation becomes vulnerable to controller node failure.

In a **distributed control** system, distributed algorithms running at each node control network operations by interacting among themselves. Processing at each node is greater than in the previous scheme, but network operations are not dependent on the continued operation of any single node.

1.1.3 Network Structure and Topology

The nodes of a network can be interconnected in a variety of ways. The particular topology chosen is important because its choice depends on factors like performance, cost, reliability, flexibility and reconfigurability of the network.

Topology and the network control strategy are also closely linked. Some of the popular topologies are:

1.1.3.1 Star Topology

A star network consists of nodes connected to a central switch via a single bi-directional link (Figure 1.1). The messages between any two nodes always have to pass through the central switch. In a star network, network operation is highly dependent upon the correct operation of the central node. Central node failure results in failure of the entire network. A dedicated path is established between the source and destination so that a single node failure does not affect the network performance or operation. Extra nodes can always be added with the capacity of the central switch being the only constraint.

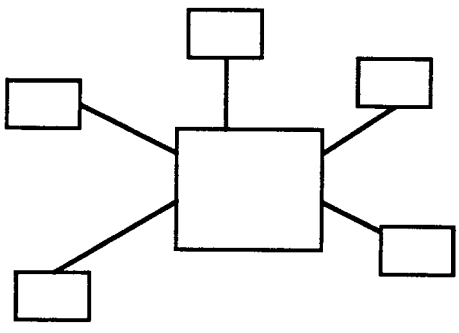


Figure 1.1 Star Topology

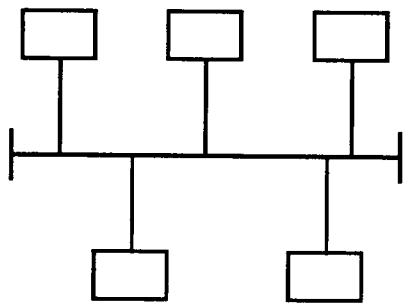


Figure 1.2 Bus Topology

1.1.3.2 Bus Topology

A bus network consists of a single transmission path for communication among the nodes (Figure 1.2). Some method must be provided to control the access of the media by the nodes. The most common techniques for this topology are Carrier Sense Multiple Access (CSMA) and token bus. A bus network operates with a single transmission medium and inherent broadcast transmission capability. Arbitration is required to access the medium. A bus network has the advantage that single node failure does not affect the network. A bus network is suitable for high-speed burst traffic.

1.1.3.3 Ring Topology

In this structure, several nodes are linked together to form the equivalent of a ring (Figure 1.3). This provides for high channel utilization while simultaneously reducing routing strategies. A message is circulated through all nodes. A ring network possesses inherent broadcast transmission capability. A disadvantage of this structure is that it is susceptible to a single link or node failure. Also, addition of new nodes would require the network to be brought down.

Among the above structures, both bus and ring topologies have some serious disadvantages. There is a degradation of performance with increased network traffic due to bottlenecks. Communications cannot be guaranteed to be private or secure. In general, however, the above presented topologies have many advantages. They offer structured and modular topologies which may be used as building blocks for more

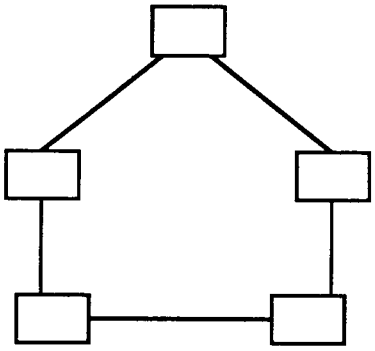


Figure 1.3 Ring Topology

complex networks. Network control is simple in all the structures, with no formal routing mechanism required. A single transmission medium is adequate.

1.2 Distributed Systems

1.2.1 Definition

A distributed system is one in which the computing functions are dispersed among several physical computing elements. These elements may be located physically close together or geographically separated.

Distributed systems usually contain multiple components of a similar nature and interconnected processing elements with no hierarchic control structure in their operation. **Separation of components** is another inherent property of distributed systems. Its consequences include the need for communication and for explicit system management and integration techniques. Separation allows the parallel execution of programs, the containment of faults to the component concerned and recovery from

those faults without disruption of the whole system. **Transparency** is defined as the concealment of separation from the user and the application programmer, so that the system is perceived as a whole rather than as a collection of independent components. The implications of transparency are a major influence on the design of the system software.

A very important language issue in the area of distributed processing is the portability of high-level languages and of programs written in high-level languages. Issues pertaining to portability are the availability and standardization of compilers, independence from the operating system, and independence of the program from internal representation of character and numeric data items.

1.2.2 Types of Distributed Systems

1.2.2.1 Loosely-coupled Systems

Figure 1.4 shows a loosely-coupled distributed system. In these types of systems, the shared resources are provided by some of the computers in the network and are accessed by system software that runs in all of the computers, using the network to coordinate their work and to transfer data between them. An example of a loosely-coupled distributed system is a single-user **workstation.** A workstation is a computer with sufficient processing power to run users' application programs. However some of the additional benefits available by the connection of a single-user workstation to a distributed system are access to shared data and resources located in other server computers via high-speed local networks.

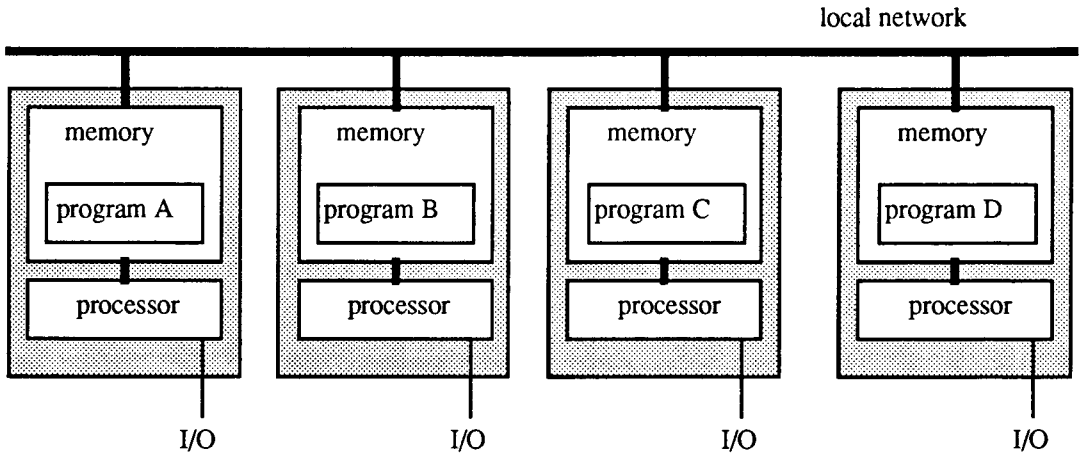


Figure 1.4 A loosely-coupled distributed system

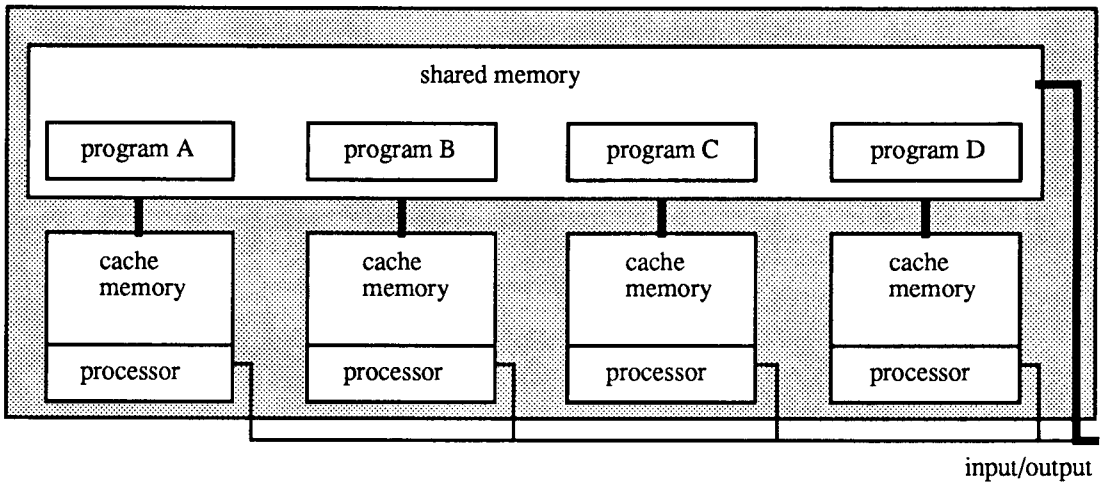
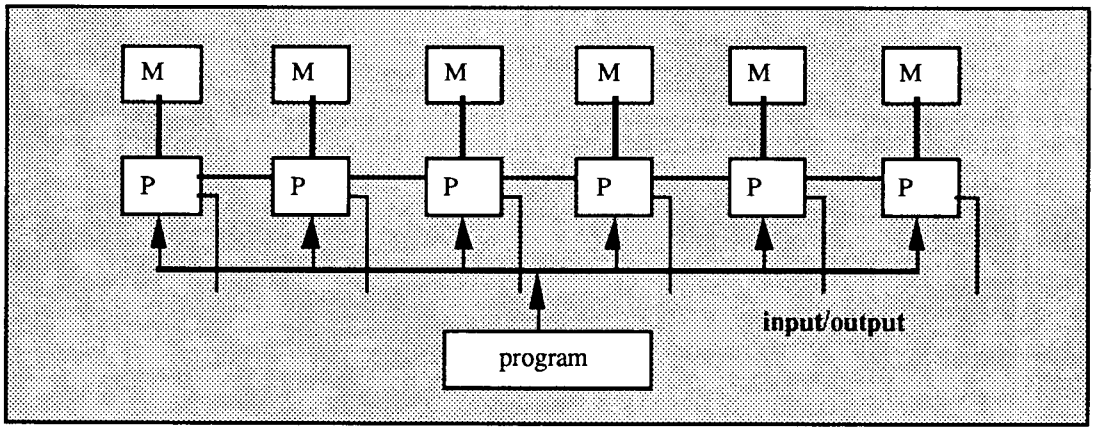


Figure 1.5 A tightly-coupled multiprocessor system

1.2.2.2 Tightly-coupled Multiprocessor Systems

Figure 1.5 shows a tightly-coupled system. This type of system integrates a number of processors under the control of a single operating system. The operating system has the responsibility of allocating processors and memory spaces to user applications and allowing them to run concurrently with the aid of a high speed connection between several separate processor/memory subsystems and a unified virtual addressing system. The use of shared memory allows the users' tasks to communicate with each other and with the operating system.



M: memory

P: processor

Figure 1.6 An Array Processor

1.2.2.3 Array Processors

Figure 1.6 shows an array processor. An array processor consists of a large number of arithmetic and logic units linked in a regular array. These can be used to perform calculations and other data intensive regular operations in parallel. A unique characteristic of this kind of system is that the entire array obeys a single stream of instructions. These Single-Instruction, Multiple-Data (or SIMD) computers are used to process large, regular sets of data at high speeds.

1.3 Data Communication

Digital data communication can be **asynchronous** or **synchronous**. A requirement of digital communication is that the receiver knows the starting time and duration of each bit that it receives.

In asynchronous transmission, data is transmitted one character (5 to 8 bits) at a time. Each character is preceded by a start bit and followed by a stop bit (Figure 1.7). When there is no data to send, the transmitter sends a continuous stop code. The

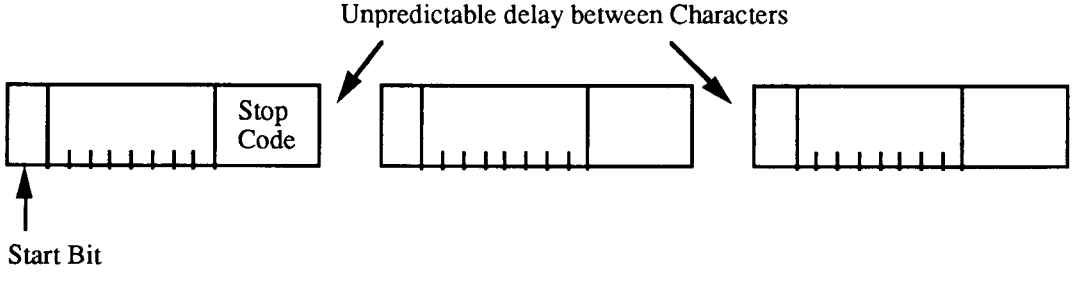


Figure 1.7 Asynchronous Transmission

receiver must have an idea of the duration of each bit in order to recover all the bits of the character. The term "asynchronous" refers to the fact that characters may be sent independently of each other at a non-uniform rate.

Synchronous transmission (Figure 1.8) is a more efficient means of communication. Blocks of characters or bits are transmitted without start and stop codes, and the exact departure or arrival time of each bit is predictable. The clocks of transmitter and receiver must be synchronized to prevent timing drift either by providing a separate clock line between transmitter and receiver or by embedding the clocking information in the data signal.

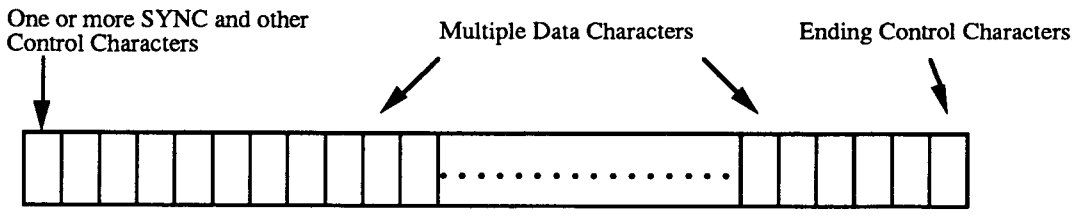


Figure 1.8 Synchronous Transmission (Character Oriented)

Another level of synchronization is required to allow the receiver to determine the beginning and end of a block of data and therefore, each block begins with a **preamble** bit pattern and ends with a **postamble** bit pattern. With character-oriented schemes, each block is preceded by one or more "synchronization characters" which are the same as the preamble. The synchronization character is chosen such that its bit pattern is significantly different from any of the regular characters being transmitted. The data plus preamble and postamble is called a **frame**.

Synchronous schemes can be character oriented or bit oriented, the latter scheme being more efficient and flexible and gradually replacing the former. Two common bit-oriented schemes, HDLC and SDLC will be discussed in section 4.1.

1.4 Benefits and Pitfalls of LANs

On the whole, LANs offer benefits as well as pitfalls. Among some of the benefits are potential for system growth since changes can be made to individual parts of the network without great impact to the remainder of the system. LANs are economically efficient since resources like expensive peripherals, data, etc. can be shared among members of the network. There is flexibility of equipment location and a user only needs a single terminal to access multiple systems.

Among the pitfalls of LANs are the fact that software and data on a network are not always compatible. Distributed data raises problems of lower security and privacy. In a network situation, there is a creeping escalation of cost with a tendency to procure more equipment than is actually needed.

1.5 Performance Issues in Computer Networks

Three measures of performance are commonly used for LANs . These measures concern themselves with performance within the local network.

• D: the delay that occurs between the time a packet or frame is ready for transmission from a node, and the completion of successful transmission.

• S: the throughput of the local network; the total rate of data being transmitted between nodes (carried load).

• U: the utilization of the local network medium; the fraction of total capacity being used.

The parameter S is often normalized and expressed as a fraction of capacity. For example, if over a period of 1 s, the sum of the successful data transfers between nodes is 1 Mb on a 10-Mbps channel, then S = 0.1. Thus S can also be interpreted as utilization. The analysis is commonly done in terms of the total number of bits transferred, including overhead (headers, trailers) bits.; the calculations are a bit easier, and this approach isolates performance effects due to the local network alone.

Results for S and D are generally plotted as a function of the offered load G, which is the actual load or traffic demand presented to the local network. S is different from G. S is the normalized rate of data packets successfully transmitted; G is the total number of packets offered to the network including control packets, such as tokens, and collisions, which are destroyed packets that must be retransmitted. G, too, is often expressed as a fraction of capacity. The more traffic competing for transmission time, the longer the delay for any individual transmission. Thus D grows without bound as more and more backlog accumulates; there is no steady-state value. S also increases with G, up to some saturation point, beyond which the network cannot handle more load.

Figure 1.9 shows the ideal situation: channel utilization increases to accommodate load up to an offered load equal to the full capacity of the system; then utilization remains at 100%. Any overhead or inefficiency will cause performance to fall short of the goal.

A more detailed analysis of network performance and other factors influencing it can be found in [STAL 90].

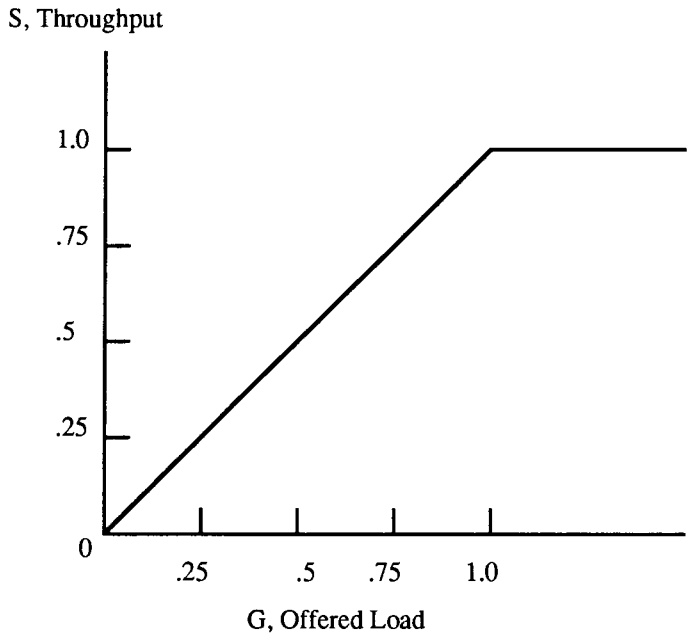


Figure 1.9 Ideal channel utilization

1.6 The Taskmaster Operating System

In addition to the classifications of networks already mentioned, most networks can be broadly classified according to their purpose as either **communication oriented** or **control oriented** networks. As the name suggests, communication networks have communication in one form or another as their main purpose. The nodes of a communication network usually follow a rigid protocol when they need to transmit/receive information in order to keep the usually heavy network traffic from various nodes organized. For example, the Ethernet network follows a communication protocol known as Carrier Sense Multiple Access/Collision Detect (CSMA/CD). The control oriented networks are intended mainly for control purposes as in industrial control, sampled data control, robotics, etc. These networks are

usually used in the real-time domain (in the sense that they have predictable time delays associated with them) for task and process control. The remainder of this project report deals with such a real-time control oriented network - the TASKMASTER.

As described by James H. Herzog and Tinggui Zhang in [HERZ 87],

> "TASKMASTER is an experimental microcontroller node which has been designed and constructed at Oregon State University to demonstrate the feasibility of the concepts previously mentioned (Task Oriented Control Structure) in performing real-time, distributed control actions."

Figure 1.10 depicts a TASKMASTER system which consists of a host terminal and one or more microcontroller units. Communication in the TASKMASTER system follows asynchronous serial protocol and any computer which supports a serial port can be used. The individual control units are "daisy chained" to one another and a command originating at the host would be received by all of them. The host in turn receives all communications from any one of the control units.

The host communicates with the desired taskmaster unit by issuing a host command packet (HCP) via its serial port. The exact format of the packet will be discussed in detail in Chapter 3 which covers the design specifications of the project. Each unit has a unique address assigned to it and is individually addressable. However, any transmission by the host computer is received by all the units. After a node determines that a particular packet is not meant for it, it does not store or actively process characters but scans for a special character indicating the start of the next packet. Standard ASCII asynchronous communication at baud rates ranging from 300 - 19200 baud can be used.

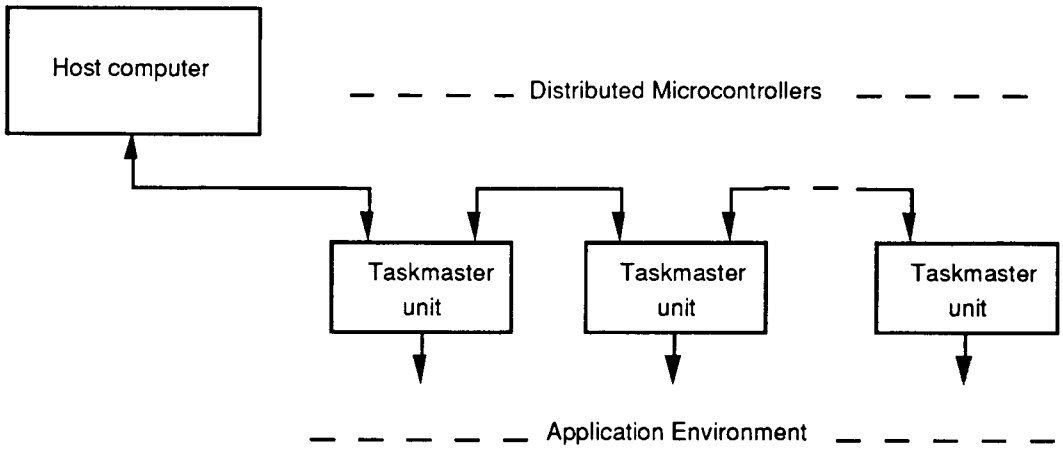


Figure 1.10 The TASKMASTER System Configuration

1.6.1 Nodes of the TASKMASTER Network

The control mechanism for the TASKMASTER system is centralized, with the host computer being the central controller. Two microcontrollers will be used individually as nodes for this study - the INTEL 8052AH and the MOTOROLA 68332. The INTEL 8052AH is an 8-bit microcontroller. It is the original member of the MCS-51 family, and is the core for all MCS-51 devices. The MOTOROLA 68332 is a 32 bit integrated microcontroller. It is based on the MC68020 and combines high performance data manipulation capabilities with powerful peripheral subsystems. These two processors represent extremes in microcontrollers with the Intel processor being more basic, inexpensive and easy to use. It is less suited for use of a high-level language than the 68332 because of fewer and less powerful addressing modes. The reason for selecting processors which are so different from each other is that there is usually no single ideal choice of controller for any control situation. Some situations require greater processing power with the cost of the processor not being a factor,

while others may need a cost efficient processor which need not be capable of intensive processing. A study involving performance comparisons of microcontrollers which are significantly different from each other with respect to processing capabilities, cost, and hardware structure would provide useful information in selecting a processor and choice of language (low or high-level) for a given control situation.

In addition to the single board computers used as nodes of the TASKMASTER network in this study, there exist a wide choice of other microprocessor and microcontroller boards available for use. The difference between a microcontroller and microprocessor is that a microcontroller is designed to be capable of independent operation by the single chip itself. It is equipped with on-chip ROM that can hold small sized programs and I/O capabilities such as general purpose ports, etc. It is suitable for control and stand-alone applications. A microprocessor in contrast usually needs additional peripherals to fully utilize its processing and hardware capabilities. It is meant mainly for use as the CPU in a larger computing system. Some of the other microprocessors and microcontrollers available are listed below according to the width of their data bus.

- 8-bits: Among the 8-bit microprocessors and microcontrollers available are the Motorola 6800, the Motorola 68HC11A4 and the Z80 from Zilog. The 68HC11A4 is a single chip microcomputer that provides sophisticated on chip peripheral functions that include an 8-channel analog-to-digital converter, a serial communications interface and a serial peripheral interface subsystem.
- 16 bits: The Z8000 from Zilog, the Intel 8086, 80186 and 80286 and the Motorola 68000 belong to this category. The Z8000 is a true 16-bit machine since the data and instruction paths are 16-bits wide. It is rich in registers. In the Intel 8086, an 8-bit

instruction path is expanded to a 16-bit external bus. The address space is extended to 24 bits in the 80286.

• 32-bits: The Motorola 68020 (and 68030), the Intel 80386 (and 80486) and the National Semiconductor 32032 (and 32332) belong to the category of true 32-bit microprocessors, with 32-bit internal and external data paths. The Intel 80486 only added on a few instructions to the 80386 but substantially increased performance.

1.7 Purpose and Importance of this Study

As previously mentioned, the TASKMASTER system is a control-oriented distributed system designed for a real-time control purpose. This study is a continuation of previous research which has been done involving the TASKMASTER system. The purpose of this study is to implement a node of the TASKMASTER system using two separate microcontrollers - the high performance Motorola 68332 and the Intel 8052 and to investigate and analyze the tradeoffs involved in using a high level language - 'C' for coding of the TASKMASTER operating system and the task routines of the network. In previous research, the operating system and tasks were coded in the assembly language of the microcontroller used. A detailed performance study will be done using performance measures defined as a part of this study to quantify the penalties paid when a high-level language is used in the case of each of the processors. The analysis will also cover the effects of different baud rates of operation and packet sizes on performance and efficiency of network operation.

This study is important because the introduction of a high level language is a valuable step in research involving the TASKMASTER system. A very significant advantage gained by using a high level language is portability and flexibility of the code. For the most part, the code is not machine hardware dependent and

consequently can be ported over to different microcontrollers, and compiled for use there after making modifications to the hardware dependent sections of the code which comprise approximately less than 10% of the overall code .

Coding of the operating system in a high-level language enables a more accurate comparison of different microcontrollers with respect to their suitability in a network situation on a common basis. This is because essentially the same high level code is being run on the different microcontrollers. Hardware differences (like bus width, etc.) and suitability of each to use of a high level language can be brought out by examining the differences in performance. High level language coding also helps make the software more readable and understandable as well as cutting down significantly on the software design and development time. The compiler library functions and routines like scanf, printf, etc. can be utilized whenever necessary instead of having to write the code for those operations. This, of course, would result in additional system overhead at run time and/or compile time.

A disadvantage of using a high-level language is the efficiency of the resulting code. The assembly code generated by the compiler is not as efficient as it would have been if coded in assembly from the start. This is true even if the compiler is efficient at generating assembly code or if the processor is well suited to use of high-level languages. Efficiency can be improved by optimizing the high-level code by coding the time-critical portions in assembly language. Careful combination of the high-level code with assembly routines and segments by an experienced programmer can yield significant improvement in performance.

Another important feature of this study is the use of the Motorola MC68332 32 bit microcontroller as a node of the TASKMASTER network. Previously, 8 bit members of the Intel MCS-51 family have been used in this research. Use of the high performance MC68332 with its powerful peripheral subsystems offers significant

scope for improvement of the control processing areas of the network. The intelligent peripheral modules include the Time Processor Unit (TPU) which provides 16 micro coded channels for performing time-related activities and the Queued Serial Module (QSM) which provides high-speed serial communications with synchronous and asynchronous protocols. The TPU greatly reduces the need for CPU intervention with its dedicated execution unit, data storage RAM and micro code ROM. The central processing unit of the 68332 excels at processing calculation intensive algorithms and can interface to a co-processor chip for floating point computation support. This makes it useful for applications such as robot control in which high performance computation is needed.

1.7.1 Experimental Procedure

Control activities involve reception, interpretation and execution of serial commands which contain information about the particular task to be performed. The TASKMASTER network previously described follows a similar procedure. The host computer transmits serial command packets which can range in size from 8 to 18 characters to the nodes of the network. A serial communication program which runs on the host computer is used to transmit a continuous stream of packets to the nodes of the network at baud rates ranging from 300 to 19200. Microsoft Quick Basic is used for this purpose in this study, although any software which is capable of serial communication can be used. The exact format and individual fields and structure of the packet will be discussed in detail in Chapter 3 which covers the design specifications of the project. A command packet contains information for node addressing, task execution priority, task number and re-queuing of the task.

Each character is received by every node of the network. Upon receipt of a character, a serial interrupt is generated and the character is stored. An address check is performed after the address field is received and only the node for which the packet is intended continues to actively process and store characters; the other nodes scan for a special character which would indicate the start of the next packet. After receipt of a complete packet, it is processed and placed on the First In, First Out task queue or executed immediately according to its priority.

The portion of time spent in the serial communication interrupt routine is of considerable importance since this part of the code is executed each time a character is received and is thus a major factor in affecting performance. This time duration will be referred to as the overhead since it could have been spent performing other useful control activities. A more detailed description of the exact measures of performance used in evaluating the network and the basis for selecting these can be found in section 2.4. The experimental procedure followed in this study can be described as follows.

The overhead includes the time spent on receiving, storing and processing task packets as well as the interrupt overhead spent on pushing the return address and other parameters on the stack, vectoring to the interrupt routine, etc. The time spent on actual task execution is not relevant in performance analysis and does not figure in calculations. The microcontrollers used in this study have on-chip timers which are used for time measurements. A set of three algorithms (which will be referred to as task 1, task 2 and task 3) which are encoded as task subroutines themselves are used for the time measurements desired.

The general operation of task 1 and task 2 is similar. Once started, they both perform the necessary timer initializations and then wait for further packets to be received by the node. A sequence of a fixed number of packets (set at 30) is sent to

the node from the host computer. From the point of view of the node, the time duration from start to finish of the stream of packets can be looked at as being made up of the overhead spent processing incoming characters and the rest of the time which represents "useful" time that could be spent performing control activities. Task 1 measures the overall time duration from receipt of the first character in the stream of packets to the last. This varies according to the baud rate of incoming packets. Task 2 measures the time taken by a simple delay subroutine to execute while the stream of packets is being received and processed. Task 3 measures this same delay subroutine but when no packets are being received. Upon completion, all three tasks transmit the time durations measured back to the host computer where they are displayed. The time durations measured by task 2 and 3 are independent of the baud rate of incoming packets and they only need to be measured once throughout the experiment.

The difference between the values returned by task 2 and task 3 yields the overhead in processing incoming packets. The difference between the value returned by task 1 and the overhead obtained from the previous step represents the "useful" time. These values can be analyzed to yield performance measures and figures of merit for the network.

The entire process is repeated for each of the microcontrollers at different baud rates ranging from 300 to 19200 baud and for different combinations of code (C and pure assembly). Two different packet types are used, with one being larger than the other in order to bring out differences in performance as a function of packet size. The microcontrollers are both run at a clock frequency of 11.0592 MHz. in order to compare results on a common basis.

1.8 Organization

This chapter has served as an introduction to Local Area Networks and distributed systems and has also covered the purpose and importance of this study. The next chapter will be a review of some literature on material more closely related to this study , and the relevance of this study to the literature. Chapter III deals with the design specifications of this project. Chapter IV will cover an analysis of the results and their implications as well as areas of further study and the conclusions of this study.

CHAPTER II

LITERATURE REVIEW

2.1 Task Oriented Control Structure

A suitable strategy of operation in control oriented systems is a "task" oriented control structure. A task can be thought of as a piece of software (or a subroutine) used to accomplish a specific action. Complex control activities can be performed by "the proper sequencing of tasks" [HERZ 87]. The operating system coordinates the execution of tasks by placing them in a FIFO (First In First Out) queue. The queue entries are moved up by one as the task at the head of the queue is completed.

There exist a wide range of inexpensive, high performance microcontrollers which are ideal for use in real-time control systems. In a system consisting of a higher order host computer and microcontroller(s), a task can be conveyed to the microcontroller by the host computer in the form of a data packet. The data packet would contain fields for the task number/name, the methods of initiating and terminating the task and its parameters (if any). A more detailed description of the exact format of the packet will be discussed in section 3.1.

A more detailed discussion of a design perspective for real-time task control in distributed systems can be found in [HERZ 83] and [HERZ 87]. The TASKMASTER system which is the focus of research in this study uses the task oriented control structure described above.

2.2 High-level Language Architecture

In the past decade, significant research has been performed to reduce a conceptual gap faced by programmers. This gap is a consequence of the fact that programmers tend to think at the conceptual level of the high-level language being used, while the hardware architecture is usually based upon totally different concepts. In [SILB 86], the authors have presented a classification of major efforts directed towards reducing the gap between programming language and architecture. In a traditional architecture, a high-level language program undergoes compilation and then is executed in the low-level machine language of the processor. The other extreme is the **direct execution architecture** in which the program is executed without any kind of translation (i.e. the high-level language is itself the machine language). Other architectures fall between these extremes.

Language directed architectures attempt to reduce the conceptual gap by tailoring the constructs of the machine code to a form that can be used by compilers for easy synthesis of high-level language constructs. Examples of this are special addressing modes (Motorola). The designers of the MC68020 made every effort to make it efficient for execution of compiled high-level code, so it can "conditionally be treated as a language directed machine" [SILB 86].

The MC68020 is a 32-bit member of the MC680X0 family. Its outstanding features are an on-chip instruction cache, an execution pipeline, a co-processor interface and (most relevant to this study) several additional addressing modes useful for high-level language data structures. Further performance improvement comes from the ability to interface to special purpose co-processors like the floating point MC68881 chip. Among the additions made to the addressing modes to provide further high-level language support are the ability to use:

- scaled indexes
- memory indirection
- 16-bit and 32-bit displacements
- preindex or postindex on memory indirect
- suppression of any element of the address calculation [MOYE 84]

2.3 Performance Analysis/Improvement in Embedded Applications

In [DONO 93], the author has listed a few basic rules by which it is possible to greatly speed up C language programs without having to resort to assembly language. The data handling procedures should be examined carefully to ensure that they are as efficient as possible. Code should be written as hardware-specifically as possible to take advantage of whatever features the hardware offers. Local variables, parameter passing, nested subroutine calls and stack usage should all be kept to a minimum.

In [MOTT 92], the author describes a technique called **Statistical Performance Analysis** which can be used for performance analysis in embedded systems. This technique utilizes a software program written by the author. Statistical performance analysis can be used to determine which modules in a program require the most execution time, which helps improve the performance of time-critical applications [MOTT 92]. The technique involves periodically sampling the program counter while a program is executing and sorting the samples according to the address range of each module after the program or the sampling is completed. The number of samples within each module is counted, and an approximate value for the relative execution time in each module is determined and displayed. The number of samples within each module is proportional to the amount of time spent in the module

executing the code, and the final count can be displayed as a percentage of the total number of samples recorded for the complete program.

2.4 Relevance of this Study to Literature

The MC68332 is based on the MC68020 which, as mentioned in Section 2.2, is suitable for use of a high-level language as it possesses a language directed architecture. Its varied and powerful addressing modes result in efficient generated assembly code. The rules mentioned by the author in [DONO 93] for speeding up C programs have been followed as much as possible in the coding of this project. Almost all variables use pointer notation which results in efficient use of memory. Most of the operating system parameters are accessed frequently and they are stored in internal RAM for fast access. Local variables and parameter passing have been kept to a minimum.

The TASKMASTER system is a broadcast type network (one in which a packet transmitted by the host is received by all nodes in the network). Further, it does not use a contention protocol when access of the medium is needed by a node. Consequently, the measures of performance listed in the previous chapter cannot be applied directly in a performance analysis of the TASKMASTER network. A few performance measures that apply more directly have been formed as part of this study and are listed below.

The procedure followed in performance analysis of the code is broadly based on "Statistical Performance Analysis" described in the previous section. Rather than actual statistical samples to determine which modules are executed most frequently, a brief inspection of the TASKMASTER code and its flow of control in a network situation is sufficient to see that the serial interrupt service routine is executed most

frequently. This section of the operating system processes the incoming characters and packets. A figure of merit is defined as a ratio of "useful" time (i.e. time not spent receiving or processing incoming characters) to the total time spent from the start to finish of a stream of packets being received by the node. This total time is the sum of the useful time and time spent on stack operations (like storing the return address before servicing the interrupt, etc.) and storing and processing incoming characters. The figure of merit is therefore a direct indication of the time spent in processing incoming packets which is the performance measure desired. The greater the value of this measure, the "better" the performance since it implies that the processor spends comparatively lesser time receiving and processing incoming characters and can devote more time to control activities.

CHAPTER III

DESIGN SPECIFICATIONS AND ASPECTS

3.1 The TASKMASTER System

As mentioned earlier in Chapter 1, the TASKMASTER system is a control network used in performing real-time, distributed control actions. A TASKMASTER system consists of a host terminal and one or more microcontroller units. Communication in the TASKMASTER system follows asynchronous serial protocol. The host communicates with the desired taskmaster unit by issuing a short command packet referred to as a Host Command Packet (HCP) via its serial port.

The general format of a host command packet is

{ AA P NN S DD DD DD DD DD } , where

AA:	2 ASCII chars	:	Address of taskmaster
P:	1 ASCII char	:	Prefix
NN:	2 ASCII chars	:	Task Number
S:	1 ASCII char	:	Suffix
DD:	2 ASCII chars	:	Data

" { " and " } " are used as delimiters for each HCP. A more detailed description of the individual fields in a packet is given below.

AA: Address of Taskmaster

Since different taskmaster units are simultaneously controlled by a single host, a command issued by the host must address a destination taskmaster. Each unit has a unique address associated with it. Address "00" is treated as a universal address to allow commands to be simultaneously addressed to all nodes.

P: Prefix

The prefix determines the starting mechanism and the priority of a task.

" : " - The task is commenced as soon as it reaches the head of the queue and the previous task in progress is completed.

" ? " - The task can be started as soon as it reaches the head of its storage queue and the currently running task is completed. However, the task does not start until the host sends a special synchronization command.

" ! " - Tasks with a ! - prefix are high priority tasks or "immediate" tasks. If a command packet with a " ! " as prefix is received, the specified task is not placed in the queue as would be done normally, but is run immediately. The currently running queue task (if any) is temporarily suspended and resumed after completion of the immediate task. The state of the interrupted task at the time of being halted is preserved while the immediate task is running. An immediate task cannot be interrupted by another immediate task (with the exception of

abort task 02). If an immediate task is issued while another immediate task is already running, the second immediate task will be ignored.

N: Task Number

The hexadecimal task number refers to the task to be run. User specific tasks can be written, in addition to the standard utility and queue-management tasks already provided by the operating system.

S: Suffix

The suffix determines the terminating mechanism associated with a task.

" . " - The task is discarded after execution.

" +" - The task is re-queued after execution and run again when it reaches the head of the queue. Re-queued tasks always remain in the queue until they are discarded by the host.

"*" - The task is re-queued a certain number of times. The number of times this will be repeated is given by the first argument. After going through the queue loop the specified number of times, the task will be discarded by the local operating system.

DD: Arguments

The argument section of the packet is meant for the purpose of passing relevant information to the task subroutines, as appropriate. The number of arguments can vary between zero and five (an even number ranging from zero to ten characters). Certain tasks may not require arguments and this field may be omitted in those cases. In the

"*" mode, the first argument is used to specify the number of times the task is to be run.

3.1.1 Organization of TASKMASTER Code

The TASKMASTER software consists of two major parts - the main routine and the serial interrupt service routine. The main routine initializes the individual taskmaster unit, opens a window for a serial interrupt to process incoming characters (if any), executes the task at the head of the task queue (if any) and manages the task queue. The serial interrupt routine receives and processes characters as appropriate. When a complete packet is received, it is analyzed. The task is placed on the task queue, or executed right away if it is an immediate task. Figure 3.1 shows the flow charts of the main and serial interrupt service routines.

3.2 Choice of a High-level Language

As explained in Section 1.3, the use of a high-level language when used to program microcontroller systems offers the advantages of readability and portability. The price paid for choosing a high-level language over the assembly language of the processor in question is the increased code generated by the compiler, and the resulting loss in efficiency of execution of the program. The choice of high-level

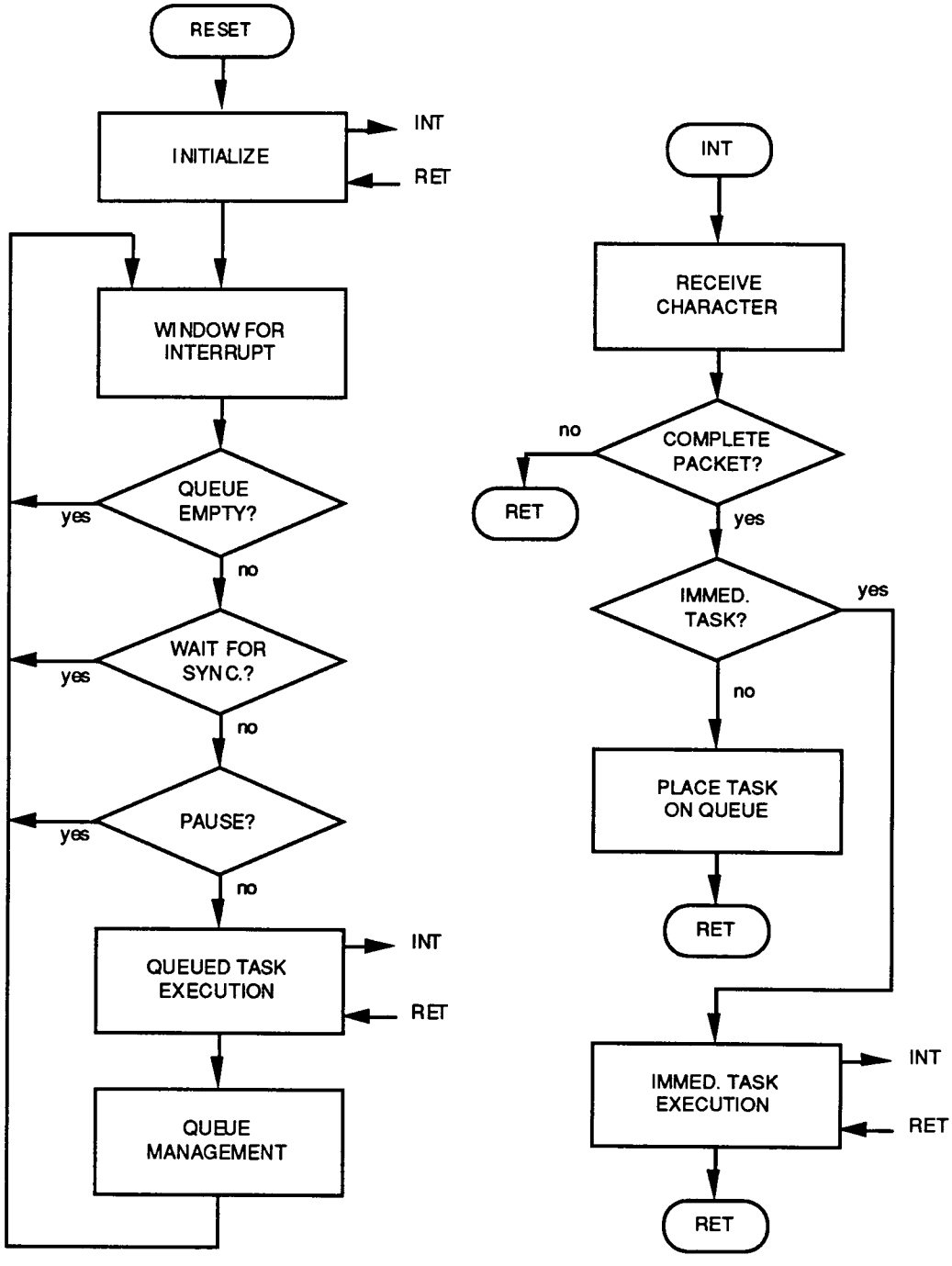


Fig. 3.1 Main and Serial Interrupt Routines of the TASKMASTER Operating System [HERZ 87]

language is therefore important. A language which has a high degree of readability, which can access the features of the processor to a greater extent and which yields comparatively efficient generated assembly code would be a good choice. Some high-level languages are "higher" than others - "it depends on the extent to which the programmer is insulated from the workings of the processor."[PETZ 88]. The high-level language chosen for this project is C, for the reasons explained below.

C is a powerful and structured language which allows programmers to construct complicated operations with a minimum of commands. C has the unique quality of combining different levels of programming abstraction from the machine code level to the highest level of applications processing. C can be thought of as one of the "lowest" of the high-level languages; the flexibility of the language makes it easy to duplicate many assembly language functions which was an important consideration in this project since the original TASKMASTER operating system was written in assembly. C is rich in data types, supporting long and short, signed or unsigned integers, floating point variables, etc. It supports pointers to all of its variable types, and pointers to variables are efficient users of memory.

3.3 Microcontrollers and C Compilers used

A brief description of the microcontrollers used as TASKMASTER units and the C compilers used in compiling their code is given below.

3.3.1 Motorola MC68332

The MC68332 is a 32-bit integrated microcontroller, combining high-performance data manipulation capabilities with powerful peripheral subsystems. The intelligent peripheral modules include the Time Processor Unit (TPU) which provides 16 micro coded channels for performing time-related activities and the Queued Serial Module (QSM) which provides high-speed serial communications with synchronous and asynchronous protocols. The TPU greatly reduces the need for CPU intervention with its dedicated execution unit, data storage RAM and micro code ROM. Two kilobytes of fully static standby RAM allow fast two-cycle access for system and data stacks and variable storage. The Central Processing Unit (CPU32) is upward compatible with the M68000 family that excels at processing calculation-intensive algorithms and supporting high-level languages.

The Compiler/Assembler system used in compiling code for the MC68332 is Release 3.0 of the TaskTools 68000 Family from Boston Systems Office/Tasking (BSO). Among the features supported in this release of the compiler are In-line assembly code capability, support for PC-relative calls, and several code quality improvements. Support for some features of ANSI C that were not supported in earlier versions has been provided.

3.3.2 Intel 8052AH

The Intel 8052AH is an 8-bit microcontroller which belongs to the MCS-51 family. It has 256 bytes of on-chip data RAM, 16 of which are bit-addressable. It uses a full duplex UART for serial communications and 3 16-bit timer/counters for time

and counting related activities. The MCS-51 instruction set is optimized for control applications. It provides a variety of fast addressing modes for accessing the internal RAM to facilitate byte operations on small data structures. The instruction set provides extensive support for one-bit variables as a separate data type, allowing direct bit manipulation in control and logic systems that require Boolean processing.

The Compiler/Assembler system used in this case is Vs. 2.0, also from BSO. It is dedicated to the microcontroller architecture of the 8052AH family which means that all special features of the 8052 can be accessed in C; multiple address spaces (with full pointer support), special function registers, interrupt functions using bank switching and a number of in-line functions to access special 8052AH instructions.

Microsoft Quick Basic is the software which runs on the host computer for serial communication with the TASKMASTER units.

3.4 Design Strategy

Coding of the TASKMASTER operating system in C to run on the Intel 8052 and Motorola 68332 processors was an important part of this study. This included the operating system task subroutines and the algorithms to measure the desired time intervals for performance analysis. Portability of the high-level code was one of the most important design aspects of this project since essentially the same program is run on the two different microcontrollers (after their individual compilations). Since the code runs on different processors, it was written in as non-processor specific a manner as possible so as to minimize the changes that need to be made when porting over to the other processor. Most of the changes were restricted to the header files since the two processors have their own distinct memory maps. All operating system variables which are accessed on a regular basis are stored in internal RAM for fast access. The

task queue in which received packets are stored after processing and the scratch memory queue are too large (400 bytes each) to be stored in internal RAM; they are stored in external RAM. The program code itself is burned into EPROM.

The C compiler used in the case of the 8052 - the **CC51** - offers two ways of dealing with the separate address spaces of the 8052. One is to specify a storage type or the target memory of a pointer with the declaration of a C variable. Different storage types exist for the separate memory types e.g. "data" for direct addressable on chip RAM, "idat" for indirect addressable on chip RAM, etc. The other way is to select a memory model that specifies which memory type must be used (as default) for all C variables which do not have an explicit storage specifier. This option is especially useful since it allows the user the option of choosing the memory model which fits best to the requirements of the system (code density, amount of external RAM, etc.). The memory model chosen for this project is the **large** model in which all function parameters and other C variables are allocated in external RAM.

The C compiler for the 68332 - the **C68332** - has a different way of allocating data and variables to memory. A switch option on the command line directs the linker to read locator commands from the file loc.cmd. By default, code and data segments are allocated in memory one after another, beginning at address 0. With the aid of locator commands, code and data can be assigned to desired locations and regions of memory can be marked "reserved" if they are not to be used for code or data.

In addition to the above changes, a few more modifications need to be made to the C source files when changing processors. These are necessary because of the differences in hardware features between the processors. Accurate measurements of the execution time of certain parts of the program are a part of the project. In the case of the 8052AH, one of the on-chip 16 bit timers is used for this purpose, while the Time Processing Unit (TPU) is used in the case of the MC68332. Serial

communications are handled by the full duplex UART for the 8052AH and by the Queued Serial Module (QSM) for the MC68332. Both the above sections of the project require unique initializations and procedures for the two microcontrollers.

CHAPTER IV

RESULTS AND IMPLICATIONS

Figure 4.1 is a table which denotes the calculated figures of merit for combinations of processor, language used and baud rates. As mentioned earlier, the figure of merit is an indication of the time spent in processing incoming packets. More precisely, it is an indication of the time that could be spent performing activities other than processing incoming packets i.e. the "useful" time. It is calculated by the formula:

$$\text{Figure of merit} = \frac{\text{Useful time}}{\text{Total time from start to finish of receiving a string of packets}}$$

$$= \frac{\text{Useful time}}{\text{Useful time + Time spent in I/O for receiving and processing packets + time spent on stack operations (storing return addresses, etc.)}}$$

The figure of merit is expressed as a percentage and the higher its value, the "better" the performance as this implies a longer period of time that can be spent on useful control activities.

The following implications can be drawn from the tabulated data:

- The 8052 coded in assembly has the best performance figures. The reason for this is obvious - the entire operating system is hand-coded in assembly and hence is much more efficient than compiler generated C code. The differences are especially obvious at higher baud rates. At 9600 baud, the figure of merit is greater than the corresponding value for the 68332 coded in C by a factor of 20% and greater than the case when the 8052 is coded in C by a factor of 40%.

Baud Rate	8052 (assembly)		68332 (C)		8052 (C)	
	Packet 1	Packet 2	Packet 1	Packet 2	Packet 1	Packet 2
300	99.41%	99.46%	99.01%	99.15%	98.50%	98.60%
1200	97.63%	97.83%	96.02%	96.61%	94.19%	94.77%
2400	95.27%	95.66%	91.97%	93.23%	88.17%	89.48%
4800	90.53%	91.31%	84.06%	86.40%	75.41%	78.87%
9600	81.05%	82.61%	66.80%	72.53%	41.52%	55.45%
19200	62.06%	65.22%	-	-	-	-

Packet 1 = {XX:XX.}
Packet 2 = {XX:XX.DDDDDDDDDD}
'-' denotes saturation of node i.e. the software cannot keep up with incoming packets

Figure 4.1 Figures of Merit

• The 68332 coded in C exhibits better performance figures than the 8052 coded in C. One reason for this is the 68332's more powerful addressing modes and instruction set which are better suited to high-level language code generation than in the case of the 8052. Consider the short segments of code shown below. They show the compiler generated assembly code for a high-level "**if**" statement in the case of the 8052 and the 68332 respectively.

```
if ((*LTQCNT == 0x0000)          /* C statement for the 8052        */

78AC      MOV  R0,#0ACH          1      ; Generated assembly code
8602      MOV  R2,@R0            1
08        INC  R0                1
8603      MOV  R3,@R0            1
EB        MOV  A,R3              1
4A        ORL  A,R2              1
6014      JZ   _5                2
                                 8 x 12 = 96 clock cycles (1 machine
                                 cycle = 12 clock cycles in the 8052)

if ((*LTQCNT == 0x0000)          /* C statement for the 68332       */

286EFFFC  MOVEA.L  -4(A6),A4     6      ; Generated assembly code
4A94      TST.L (A4)             2
67000014  BEQ      L10000        4
                                 12 clock cycles
```

The first column contains the object code and the numbers in the last column are the execution times of the instruction in clock cycles. These figures are a better indication of the difference between the two processors rather than simply the number of instructions generated (7 for the 8052 vs. 3 for the 68332) since instruction execution times differ. The clock cycle figure for the 8052 is multiplied by 12 since a machine cycle takes 12 oscillator periods to execute for the 8052. The final figures arrived at are 96 clock cycles for the 8052 vs. 12 clock cycles for the 68332. Generalizations or predictions based only on figures obtained from a short segment of code should be made with caution since not all code segments would yield the same differences. The code segment just examined is biased in favor of the 32-bit Motorola processor since the "if" statement tests a value which is greater than 8 bits (the data width of the 8052's bus). The Motorola processor has an **orthogonal** instruction set

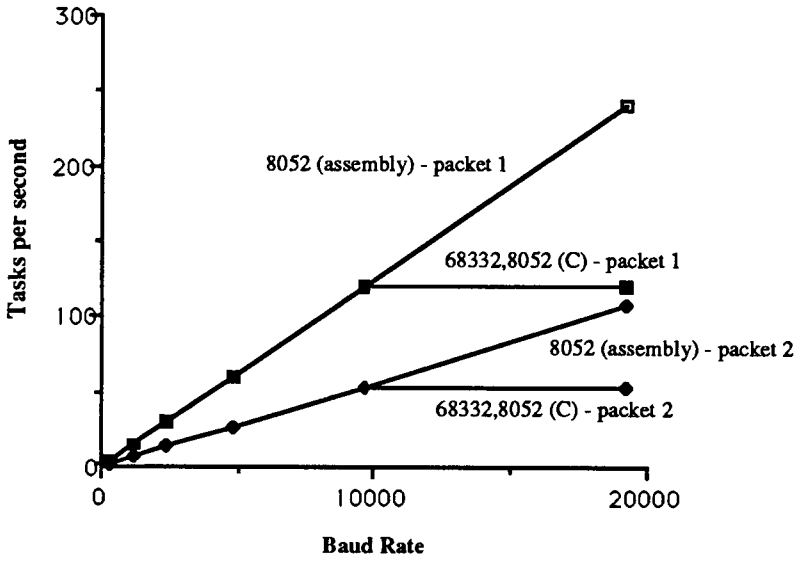
in the sense that most of its instructions can handle byte, word or long-word data operands while the 8052 can only handle byte operations. The TASKMASTER code contains some parameters which are larger than a byte and the 68332 is at an advantage when performing operations involving these parameters.

From inspection of the generated assembly code, it can be seen that the 68332's instructions are more powerful since the instruction set design allows greater flexibility and choice in addressing modes for data operands. They take longer to execute per instruction on average since an effective address calculation from the complex addressing modes takes time. The 8052's instructions are correspondingly less powerful since the instruction set only supports simple addressing modes and take fewer clock cycles to execute on average. Almost all two-operand instructions need to have a register as one of the operands while the 68332 does not have this limitation and can perform most operations on memory directly without needing to fetch the data into a register and store it back into memory. The result is an instruction set that is well suited for compiling the complex structures of high-level languages.

- Performance figures are more or less independent of packet size as can be seen by comparing the figures of merit for packet 1 and packet 2. Packet 2 contains data arguments meant for the purpose of passing relevant information to the task subroutines, as appropriate. Not all tasks require arguments and this field may be omitted in those cases. Although a larger packet implies more time to be spent on I/O, etc. there is a corresponding increase in "useful" time as on the whole a greater time is available between characters being received to do other activities. The ratio of these two quantities thus yields a figure of merit which is equal to that obtained when a smaller packet size was used.

The figure of merit does not reflect a penalty that is paid for the case when a larger command packet is used. Since it takes longer to transmit and process the larger

sized packet, lesser number of tasks would be executed on average than if a smaller sized packet (with no data arguments) had been used. This is shown graphically in Figure 4.2. which shows the tasks received by a node per second as a function of the baud rate of incoming packets.



Packet 1 = {XX:XX.}
Packet 2 = {XX:XX.DDDDDDDDDD}

Figure 4.2 Tasks/second vs. Baud Rate

The exact number of tasks received is independent of the processor or language used which is the reason that the curves coincide for a given packet size for baud rates till 9600 baud. After this point, the nodes coded in the high-level language are saturated since the processors cannot keep up with the incoming characters and

the tasks received level off to a constant figure. The case for when the 8052 is coded in assembly is more efficient and can keep up with the characters.

The number of tasks received by the node per second are significantly less when a larger sized packet is received and this implies a loss in efficiency. However, it is not always possible to control the size of the command packet at will since certain control tasks require the information in the data arguments section of the command packet and cannot be executed without this field.

- Figure 4.2 also yields some important information which is useful in selecting a suitable baud rate of operation. A high baud rate would result in a greater number of packets being received by the node per unit time and therefore improved efficiency. However, as can be seen from the earlier tabulated results, higher baud rates result in a lower figure of merit since a greater amount of time is spent processing incoming packets rather than on executing tasks which might already be in the task queue. A factor that would help resolve this contradiction is the status of the task queue. If the task queue is empty or only partially full, a higher baud rate would be more suitable since this would allow the queue to fill up. If, on the other hand, the queue is operating close to its capacity, a lower baud rate or lower rate of packet transmission should be chosen to allow more time to be spent by the node on processing the tasks already in the queue.

- Figure 4.3 is a graphical representation of the results tabulated earlier in Figure 4.1. It can be observed that the 8052 coded in assembly has a comparatively steadier figure of merit with increase in baud rates than the other two cases. This is because as baud rate increases, the large number of characters coming in cause any differences in performance at lower baud rates to be magnified correspondingly since greater time needs to be spent to process characters.

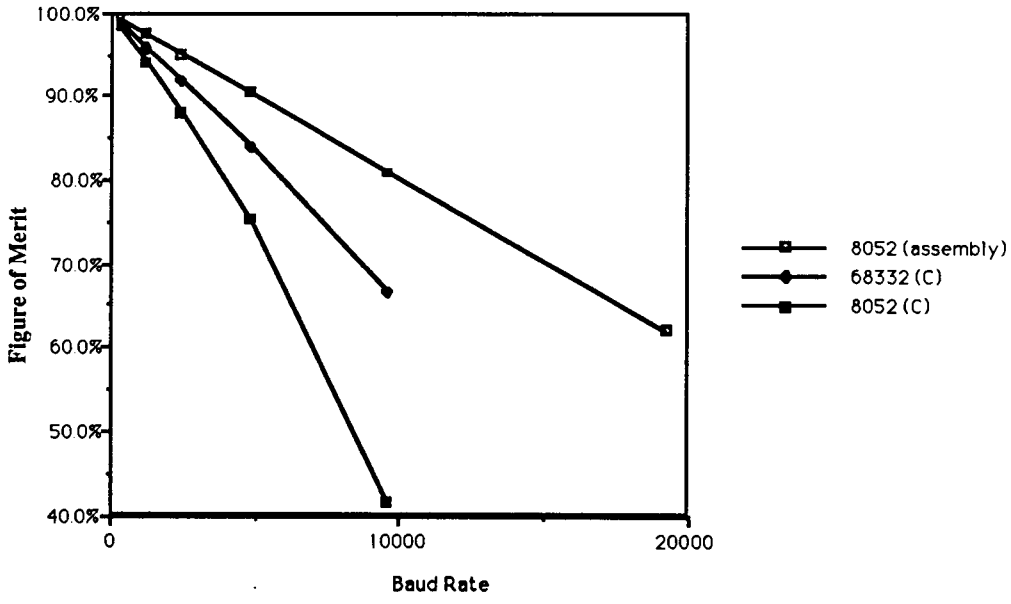


Figure 4.3 Figure of Merit vs. Baud Rate

• The interrupt overhead associated with each processor is a significant factor in performance analysis. Upon occurrence of an interrupt, the 8052 pushes only the Program Counter onto the stack, while the Motorola processor pushes the status register in addition to the PC. In contrast, the actual interrupt service routine for the 8052 needs a few more instructions than the 68332 in order to test if the interrupt is a transmit or receive interrupt and to clear the interrupt flag.

• Figure 4.4 is a representation of generated object code sizes. The 68332 and 8052 when coded in C result in almost equal sizes of object code. There is a tradeoff here between fewer resulting assembly instructions being generated for the 68332 vs. greater encoding for these instructions resulting in an almost equal object code size. The 8052 when coded in assembly results in the lowest object code size, as expected.

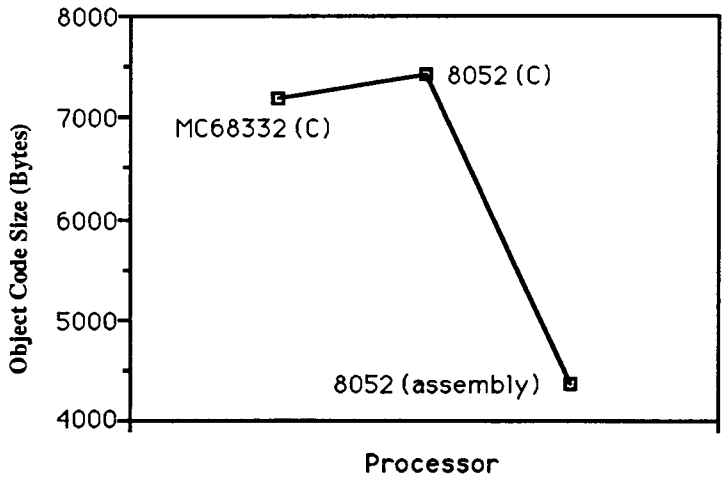


Figure 4.4 Object code size vs. Processor

4.1 Performance and Throughput Improvement

The serial interfaces of both processors used in this study can only receive a character at a time without the ability to buffer or store multiple characters received without CPU intervention. This is a limiting factor of performance since the ability to receive multiple characters would allow the processor to perform other "useful" tasks until an entire packet or multiple packets were received, rather than having to be interrupted each time a character was received. The 68332 processor possesses a Queued Serial Peripheral Interface with a memory queue for transmit and receive data which would allow multiple characters and packets to be stored before servicing. The 8052 does not possess such a feature, but the Intel 8044 microcontroller has a data RAM for multiple character reception capability. These processors will be discussed

more in detail below. They offer solutions to the above mentioned limiting factor on performance.

The asynchronous RS-232 form of communication used in this study is also a significant limiting factor of performance since the maximum data rates that can be achieved are limited to 19200 baud. An alternative form of communication which offers higher data rates is the synchronous High Level Data Link Control (HDLC) or Synchronous Data Link Control (SDLC) which is a subset of HDLC. The major advantages of SDLC/HDLC over the asynchronous communications protocol are simplicity, efficiency (higher data throughput) and reliability which is achieved by a Frame check sequence and Frame numbering. A typical HDLC/SDLC frame consists of an opening flag, an address field, a control field, the data field, a frame check sequence and a closing flag.

Most processors which support synchronous communication capabilities also have serial buffering capacity to store multiple data fields. As previously mentioned, the 68332 possesses the Queued Serial Peripheral Interface (QSPI) which is a full-duplex synchronous serial interface with a queue for transmit and receive data. The QSPI also has other useful features like a programmable queue for up to 16 pre-programmed transfers without CPU intervention and a continuous transfer mode of up to 256 bits.

The Intel 8044 microcontroller integrates onto a single chip the 8051-core with an intelligent and high performance Serial Interface Unit (SIU) which is designed to perform serial communications with little or no CPU involvement. The SIU supports data rates up to 2.4 Mbps and the HDLC/SDLC protocol and has a data RAM.

HDLC's efficiency is questionable when used in control applications [GRAU 80] if the network bandwidth is a limiting factor and needs to be conserved. Process

control messages usually tend to be relatively short and the HDLC overhead becomes a problem with regards to efficiency.

4.2 Areas of Further Study

- As discussed above, use of synchronous protocols would yield advantages like improved data rates, etc. HDLC and SDLC, in particular, are options for synchronous communication protocols worth investigating. The MC68332 would be well suited for this since it possesses the QSPI which has capabilities for CPU independent synchronous communication and also a queue for transmit and receive data.

- A hybrid mixed control network made up of MC68332's and 8052's could be investigated. Each of these two processors' have advantages and disadvantages when used as nodes of the TASKMASTER network. It is not necessary to use the more expensive MC68332 if the control activities to be performed are simple and do not call for intensive calculation or powerful, independent peripherals. In this case, the less expensive and simpler 8052 will suffice. Through judicious sequencing of tasks divided selectively between the processors, it should be possible to achieve efficient system performance, throughput and cost.

- Hand optimization of the compiler generated code, and/or rewriting of time critical portions of the operating system in pure assembly would yield better performance. Both the C compilers used allow the use of in-line assembly code mixed with the high-level code as well as calls of assembly routines from within the C code. The serial interrupt routine should be concentrated on since any optimization here would increase performance significantly since this part of the code is executed the most frequently.

- The MC68332 is a high-performance, integrated microcontroller with powerful peripheral subsystems which can be used to advantage in an industrial control environment. The sophisticated Time Processor Unit (TPU) provides optimum performance in controlling time related activity. It has a dedicated execution unit which greatly reduces the need for CPU intervention. It would be worthwhile to exploit these and the other high-performance peripheral subsystems according to individual control needs.

4.3 Conclusions

As part of this study, two different microcontrollers - the Motorola MC68332 and the Intel 8052 were used to implement nodes of the TASKMASTER network. The operating system of the network that runs on each of the nodes was coded using a high-level language - C. The task routines and algorithms for the network were also coded in C, resulting in an overall code size of approximately 700 lines. A performance analysis was done using measures formulated as part of this study to determine suitability of each of the processors to use of a high-level language and the penalty paid when a high-level language is used instead of the assembly language of the processor. The analysis also included performance measures of the processors at different baud rates and for different packet sizes.

The MC68332 is a more powerful microcontroller with better performance figures than the 8052 when they are both coded in C. Pure assembly coding of the 68332 processor was not performed as part of this study, but it can be expected to yield better performance than the 8052 coded in its assembly language. The 68332 is ideal for use in a control network on account of its many independent peripherals. This allows the CPU to continue with control activities while the peripherals handle

operating system activities such as receiving incoming characters and time measurements on their own without CPU intervention. The choice of processor is ultimately application dependent. The Intel 8052 microcontroller is less expensive and simpler to use and it may suffice for applications which do not require great processing power or intensive data manipulation and for which high performance is not critical.

Regarding the choice of language, for time-critical applications, assembly language coding would yield the best results in exchange for greater software development time required. However, the use of a high-level language is strongly recommended if the application is not performance-critical or if the slight performance penalty paid is not a significant factor. Frequently executed portions of the software can be coded in assembly for improvement of performance. Both the compilers used in this study support in-line assembly inter-mixed with the high-level code. The software development time for a high-level language can be very much lesser than for an assembly language, particularly in the hands of an experienced programmer well acquainted with the processor architecture and compiler used. A high-level language is very readable in general and easier to debug and revise at a later time, should the need arise.

CHAPTER V

BIBLIOGRAPHY

COUL 88 George F. Coulouris and Jean Dollimore, *Distributed Systems - Concepts and Design.* Addison-Wesley Publishing Company, Inc., 1988.

DONO 93 John Donovan, "Careful programming lets C replace assembler in fast embedded applications," *EDN Journal*, vol. 38, no. 8, pp. 81-88, April 1993.

GRAU 80 Maris Graube, "Proway -- A Local Network for Process Control," *IEEE Corporate Interface Engineer*, pp. 313-316, 1980.

HERZ 83 James H. Herzog, "A design perspective for real-time task control in distributed systems," *IEEE transactions on Industrial Electronics*, vol. IE-30, No. 1, pp. 46-51, February 1983.

HERZ 87 James H. Herzog and Tinggui Zhang, "A design methodology for distributed microprocessors in real-time control applications," Paper presented at Second International Conference on Computers and Applications, Beijing, People's Republic of China, June 24-26,1987.

MOTT 91 Fred Motteler, "Statistical Performance Analysis: looking for quality time," *Dr. Dobb's Journal*, vol. 16, no. 12, pp. 68-79, December 1991.

MOYE 84 Doug MacGregor, Dave Mothersole and Bill Moyer, "The Motorola MC68020," *IEEE Micro*, pp. 101-118, August 1984.

PETZ 88 Charles Petzold, "The C Mystique," *PC Magazine*, vol. 7, No. 15, pp. 92-108, September 13, 1988.

SILB 86 A. Silbey, V. Milutinovic and V. Mendoza-Grado, "A survey of advanced microprocessors and high-level language computer architecture," *IEEE tutorial on advanced microprocessors and high-level language computer architectures*, pp. 118-141, 1986.

STAL 90 William Stallings, *Local Networks.* New York: MacMillan Publishing Company; London: Collier Macmillan Publishers, 1990.

TANE 88 A. S. Tanenbaum, *Computer Networks*. Englewood Cliffs, NJ: Prentice-Hall Inc., 1988.

68332 *MC68332 User's Manual* - Motorola. Motorola Literature Distrubution, Arizona.