Real Time Multitasking System Application Incorporating VRTX

by

Pradyumna Kumar Misra

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Electrical Engineering

APPROVED:

Dr. Charles E. Nunnally, Chairman

[illegible]
Dr. Joseph G. Tront

Dr. Richard O. Claus

July 8, 1987

Blacksburg, Virginia

Real Time Multitasking System Application Incorporating VRTX

by

Pradyumna Kumar Misra

Dr. Charles E. Nunnally, Chairman

Electrical Engineering

(ABSTRACT)

The real time multitasking systems are becoming increasingly popular for control and monitoring functions typically encountered in industry as well as day to day life. They have to manage adequately many concurrent processes or tasks, each of which is sequential in nature. The concurrency is achieved by running asynchronous tasks at different speeds and providing for communication and synchronization. In order to fully exploit the power and capabilities of today's sophisticated microprocessors and to provide a programming methodology for structuring real time applications a real time multitasking operating system becomes critical.

The VRTX/86 series of components from Hunter and Ready which include a real time executive, an input output executive and a file management executive provide multitasking capabilities to a microprocessor based system. This thesis deals with VRTX in great detail. A real time multitasking application was chosen in order to demonstrate the concepts of multitasking and provide a design method for real time multitasking systems. A vehicular data acquisition system was chosen as an example to achieve it. The main functions of such a system are acquisition of information from sensors, recording in a predefined format and storing it for a sufficiently large time for a later analysis.

This thesis presents the whole process of development of such a system based on a 80C88 microcomputer board and VRTX kernel. The main activities were designing of a board support package which links the kernel to hardware. The application software was then developed to exploit the features provided by the integrated system with VRTX ported onto it.

Acknowledgements

I wish to thank Dr. Charles Nunnally for his continuous support, encouragement and guidance without which this thesis would not have been possible. I would also like to thank Dr. Joseph G. Tront and Dr. Richard O. Claus for being on my committee and providing me with useful suggestions. I also wish to thank my wife for being so understanding and supportive.

Table of Contents

List of Illustrations

List of Tables

1.0 Introduction

The continuing growth in sophistication and complexity of microprocessor systems demands an increasing amount of system software support to efficiently perform the basic system operations such as input/output, timing, interrupt handling and data management. The most important component of system software is the Operating System. Until recently not many operating systems were commercially available and hence microprocessor users were forced to design their own or do without it. These user designed systems are termed as Executives and are normally tailored to suit the specific needs of a given system.

The function of an operating system is to provide the user with a Virtual Machine so that he is shielded from the machine specific hardware details. This is accomplished by providing the user with a set of commands and services which can be used to achieve desired results. The operating system implements a logical to physical mapping transforming the logical concepts of information processing into the physical concept of information processing at the machine level [42].

Almost all real time systems utilize sensors for acquiring information from the world outside them and respond to a situation in a predetermined fashion within a fixed time. Consequently the demands of software development in real time environment are drastically different than those for normal computer application. The primary area of interest in such systems is the process being controlled which can present to the computer a new situation very often and will need immediate attention. Since it is not possible for a software designer to take into account all possible scenarios, except in very small systems, there is a critical need for versatile software tools to address this problem.

The Real Time Executives or Operating Systems are the most important of those tools and services, that present to user an entire new view of the system which is largely independent of the hardware intricacies. They also provide an environment conducive to real time software development by providing a virtual machine to designer offering many critical system services. Today there are many companys, especially the vendors of microprocessors, which sell real time executive kernels which can be ported to a given hardware system. VRTX from Hunter and Ready, iRMX from Intel, MTOS from Industrial Programming are few among them.

This thesis deals with following issues.

- Develop an understanding of real time multitasking systems.
- Port the VRTX/88 kernel from Hunter & Ready onto a single board microcomputer system.
- Develop an application utilizing the services of VRTX/88.

A vehicular data acquisition system from Red Pine Associates was analyzed to gain valuable insight into a real time system. A comprehensive literature survey was undertaken to understand the concepts of multitasking and the way they relate to performance and needs of real time systems. The most important part of this thesis deals with porting a VRTX/88 kernel from Hunter and Ready onto a microcomputer board acquired from Microsystems International. Subsequently some functions of the vehicular data acquisition system were implemented using VRTX/88 services to demonstrates the versatility of multitasking environment.

It is appropriate to mention that this thesis seeks to consolidate and expand upon the work already done in this area by Shanker [35], Reddy [32] and DeBrunner[5]. The main area of concentration in case of Shanker [35] was understanding, troubleshooting and installation of automatic passenger counter system. The automatic passenger counting systems are considered to be the forerunners of vehicular data acquisition systems explored and analyzed by Reddy [32] and in this thesis. Shanker [35] also suggests a hardware design to implement a multitasking automatic passenger counting systems using more powerful processor. His thoughts on this issue were carefully examined while a choice was made to procure off the shelf hardware for this thesis work.

The work done by Reddy [32] provides a good platform to start designing and building a system using VRTX silicon software components. Even though his thesis concentrates on providing multitasking on an IBM PC it provides a working example for a start. Reddy [32] also provides a theoretical design for a vehicular instrumentation system in his thesis which served as basis for system specifications and guidelines in the present case. The system considered in this thesis is however more complex because of enhanced system requirements and more elaborate sensor mechanisms. His work was continually referred to during intermediate stages of task identification and design for the new system.

Another useful source of information for this thesis was the thesis work of DeBrunner [5] which concentrated heavily on software design aspects of a multitasking system. This material served as the starting point for software design of present system and was referred to during phases of data flow graph design and functional decomposition. It also provides exhaustive and balanced information regarding other silicon software components from Hunter and Ready like IOX, FMX and TRACER.

As mentioned earlier this thesis work attempts to consolidate and draw upon the work mentioned above as well as to enlarge it. The approach was to try out some or as many as possible ideas presented in these works by designing and building a multitasking system from the basic hardware up. The first phase of this thesis was devoted to literature survey and study of this material along with large amount of documentation supplied by Hunter and Ready.

The second phase consisted of analyzing a vehicular data acquisition system hardware and software as well as study of similar systems to arrive at system specifications. The third phase of the thesis is devoted to the activities connected with identification and selection of appropriate hardware, preliminary system design, field input and operator input interface decisions and identification of other hardware and software tools that may be required.

The fourth phase was concerned with getting the hardware to running with VRTX installed and all boards connected in a bus system. The most important activity in this phase was porting of VRTX to the hardware by developing a comprehensive board support package. The final phase dealt with design, development and testing of application software. It was developed and tested in small increments. Note that not all of the functions were implemented due to various constraints. The functions realized clearly demonstrate and illustrate the process of developing a real time multitasking system from the specification level. It also highlights the advantages of using multitasking approach and versatility of real time executives.

2.0 Operating System Concepts

An operating system may be viewed as an organized collection of software that controls and sequences the execution of the user programs in a computing environment. It is responsible for managing the system-wide resources like cpu time, input/output devices, files etc. It has to keep track of status of all of the system resources and provide regulated access to contending programs. Other important function would include resolving conflicts in case of simultaneous requests for accessing the same resource, error detection and handling, maintaining the integrity of files and other information resources.

An operating system normally comprises of three components namely a command line interpreter, a nucleus or kernel and a set of input and output device drivers [21].
This has been shown in Figure 1 on page 6. The main objective of the **command interpreter** is to implement a human interface so that users can interact with the system. The **Kernel** or nucleus is the heart of an operating system and plays very important role in almost all of the resource management decisions. It provides a logical interface between the hardware and the software running on it. The **I/O drivers** are responsible for controlling the peripheral devices in accordance with kernel directives. They are very much hardware dependent and shield the user and the kernel from actual hardware.

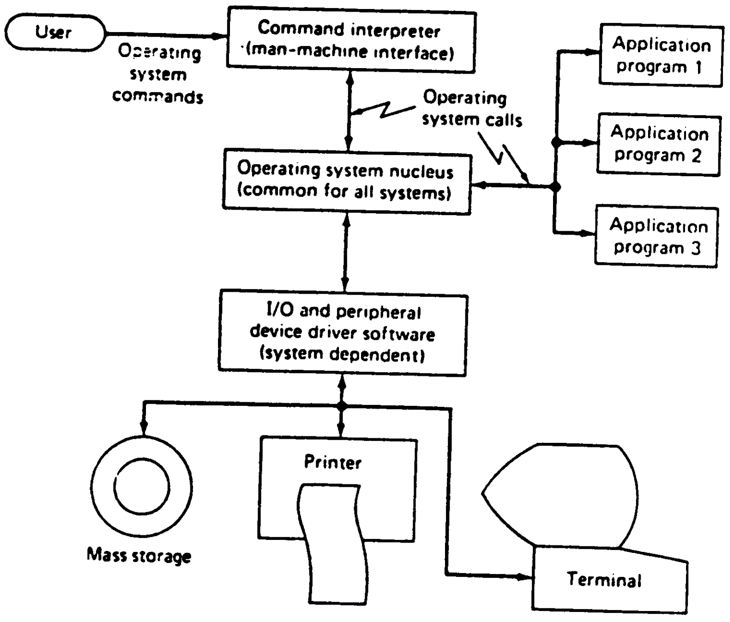

User
Operating system commands
Command interpreter (man-machine interface)
Operating system calls
Operating system nucleus (common for all systems)
Application program 1
Application program 2
Application program 3
I/O and peripheral device driver software (system dependent)
Printer
Mass storage
Terminal

Figure 1. Basic Elements of Operating System [21]

2.1 Operating System Services

Operating System attempts to provide a user friendly and easy to use environment for program execution by making available certain services to programs and users. The specific services differ from system to system but they can be classified under following categories [29].

Program Execution	The system must be able to load the program in the main memory and able to run it. The execution must terminate normally or otherwise.
Input/Output Operation	The system must be able to provide some means to interact with the computer, that is, it should provide facilities to read from keyboard, write to file or printer etc.
File Systems	Most of the systems with sufficiently large memory provide a file system to the users which hides the details of memory access and usage from the programs and users. The system allows creation, manipulation and maintenance of files which are basically logical constructs.
Logical I/O System	Since application programs use I/O functions by calling the operating system, they can be isolated from the physical details of the peripheral devices [21], instead they deal with logical devices. The system keeps a table indicating the mapping of logical devices to actual devices.

Multitasking	Some operating systems allow simultaneous execution of more than one program. This is accomplished by allowing each program to run in short bursts of time giving an illusion of concurrency. A scheduler and some scheduling algorithm is used to support it.
Resource Management	Multitasking systems must have a means to allocate system resources such as memory and I/O devices among the programs competing for them. Sophisticated memory management schemes like paging and segmentation are used to ensure efficient memory utilization. I/O devices too are carefully managed using techniques like spooling.
Message Passing	The ability to transfer information from one program to another is very critical in multitasking systems. It is particularly important in real time systems where control software is divided into a number of tasks which have to co-operate in order to achieve the desired results.

These services are provided via system calls and or system programs. The basic level of services related to process management, device and file manipulation and information maintenance are available to user via system calls. However most operating systems are augmented with a large collection of system programs designed to provide services pertaining to routine and frequently recurring activities. File copying, directory maintenance and programming language support are some of such services.

The operating systems are event driven programs and events are always signalled by interrupts or traps, thus they are interrupt driven [29]. In response to an interrupt or trap the operating system saves the present context, analyzes it for the type of service required, provide that service if appropriate, return to point of interruption, restore the cpu status and resume. A general flow diagram of an operating system is given in Figure 2 on page 10.

2.2 *Type of Operating Systems*

There are many criteria which can be used to classify different type of operating systems in existence. These can be broadly divided into uniprocessing and multiprogramming or multiprocessing systems. The multiprogramming systems can further be divided into multitasking, multiuser, time-sharing and foreground/background systems. The batch processing systems are prime example of uniprocessing systems. These are briefly summarized below.

2.2.1 Batch Processing Systems

A batch processing system requires availability of the program, data and system commands before processing the job. There is little or no interaction with the user once program starts execution. It is good for non time critical programs like payroll processing which require little user interaction and consume lot of time.

The jobs are handled on first-come-first-serve basis except in certain special cases when short jobs are allowed to run first. Memory is divided into two parts, first the permanent store holding the resident part of operating system and the second the transient store which holds

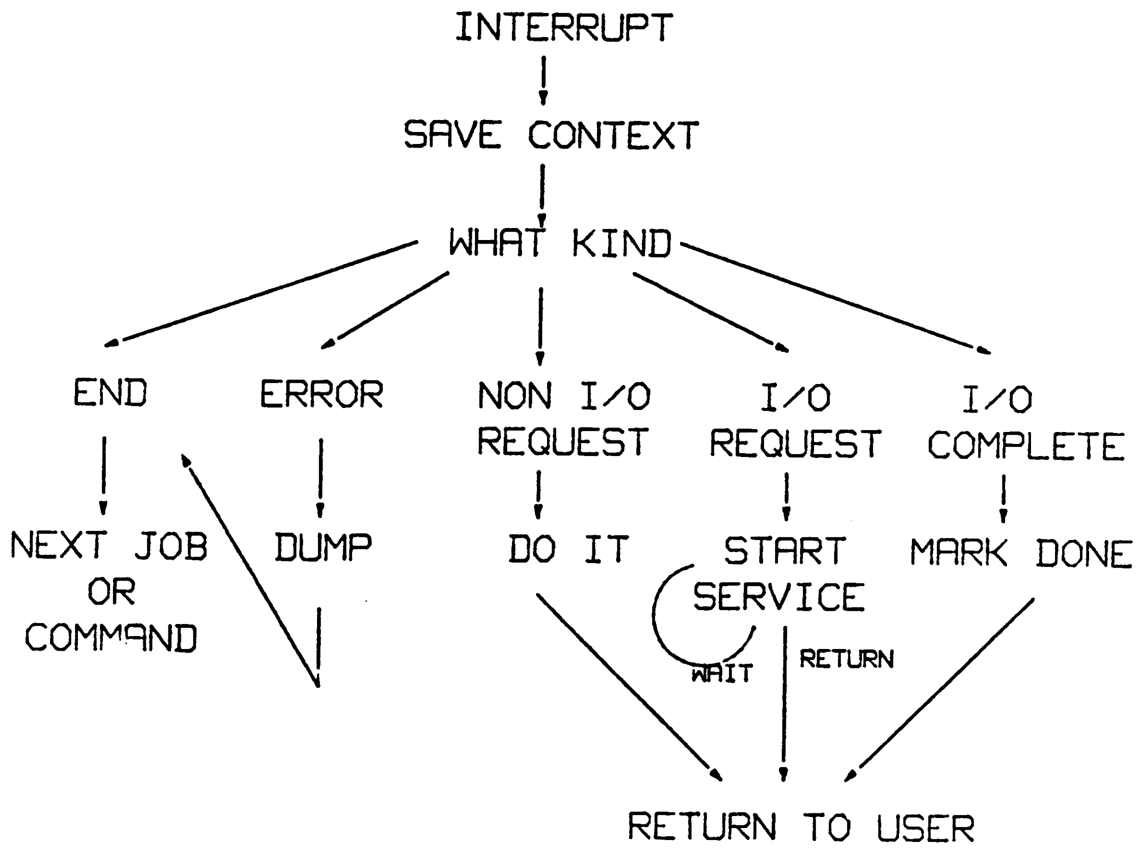


Figure 2. General Flow Diagram of Operating System

the code and data concerning currently running program. Since only one program runs at a time there is no need for sophisticated device management schemes.

Batch systems do provide some improvement over serial processing but are too simplistic to make efficient and productive use of computer hardware. They are absolutely non-candidates for real time systems where the time is always at premium and resources scarce.

2.2.2 Multiprogramming Systems

Multiprogramming systems are characterized by their ability to support multiple programs concurrently. An instance of a program in execution is called a process or task. Apart from supporting concurrent execution of multiple processes these operating systems also allow instructions and data from two or more disjoint processes to reside in the main memory simultaneously [23]. The fundamental characteristic of any multiprogramming system is the presence of more than one simultaneously active programs that compete for various system resources. All multiprogramming operating systems make use of multitasking but converse is not automatically implied.

2.2.3 Time Sharing Systems

Timesharing systems combine the multiprogramming approach with interactive computing. A time shared operating system uses cpu scheduling and multiprogramming to provide economical use of system [23]. By rapidly switching the processor between different tasks it gives the user illusion of having the machine to himself and tries to provide equitable sharing of resources to sustain it.

In order to achieve 'fairness' in resource allocation timesharing operating systems often employ time-slice or round-robin scheduling. In this approach programs are executed with rotating priority that increases during waiting period and drops after service is granted [23]. Since each process gets only a window of cpu time no task is allowed to monopolize the processor. Allocation and deallocation of memory and input/output devices is done in a manner that preserves the system integrity and yields good performance.

2.2.4 Real Time Systems

Real Time operating systems are useful in situations where a computer is required to recognize and process a large number of events, most of which are external, within rigid time constraints. Industrial process controls, telephone switching equipment, flight controls and real time simulations are some of the many applications which rely heavily on this kind of operating systems to provide reasonably economic and satisfactory services.

The most important objective of real time operating systems is to provide quick event-response time and features like efficient resource utilization and user friendliness are relegated to lower level of concern. Almost all real time operating systems employ multitasking in some form, however it may be absent in some very simple systems. The tasks are scheduled for execution independently of each other.

Normally there will be a dedicated task in the system assigned to handle a particular event. Each task in the system is given a priority which is based on the relative importance of the event it is designed to service. The system maintains a queue of ready to run processes in the system and the processor is allocated to the highest priority task in the queue. Moreover the higher priority tasks are allowed to preempt the execution of lower priority tasks. This is referred to as preemptive scheduling and is used in majority of real time systems.

Since most of the processes are permanently resident in the main memory, in order to provide quick response, memory management becomes fairly simple. The device management however is highly critical as the system has to support multiple system calls from various tasks in addition to performing routine I/O operations like buffering and interrupt management.

2.3 Task Management

The concept of a task or process is implicitly or explicitly associated with all computing environments that support concurrent execution. In essence a process or task is an instance of a program in execution and is the smallest schedulable entity [23]. Most of the systems provide following functions for task management.

1. Creating and removing tasks.

2. Controlling the progress of a task, that is, ensuring that a logically enabled process makes progress towards completion.

3. Handling exceptions arising during program execution.

4. Providing inter-process communication.

Once the system has been started there must be a least one running task which can accept requests from users and initiate their processes [37]. Upon receiving a request to activate an executable program the operating system responds by creating a process. Once created it becomes eligible to compete for system resources.

For example when a user invokes an editor program the system creates an editor process and schedules it for execution. This editor process will now accept and process the user commands. Now if another user invokes the editor the system creates another editor process for the second user by using the executable editor program as template. When user exits the program this process winds up and alerts the operating system which in turn completes the housekeeping and deletes the process. At the time of creation each task is assigned certain attributes that assist management by the system.

A given task may exist in any one of the possible four states. illustrates the process state transition diagram. The four permissible states are ;

Ready	A task is said to be in this state if it possesses every resource except the cpu for execution.
Running	A task possessing all required resources including the cpu is said to be in this state, that is, it is currently running.
Suspended	A task is said to be in this state if it needs some resources or messages in addition to cpu to start execution.
Dormant	A task is said to be dormant if the cpu is not aware of its existence that is either it has not yet been created or has been deleted.

Please see Figure 3 on page 15.

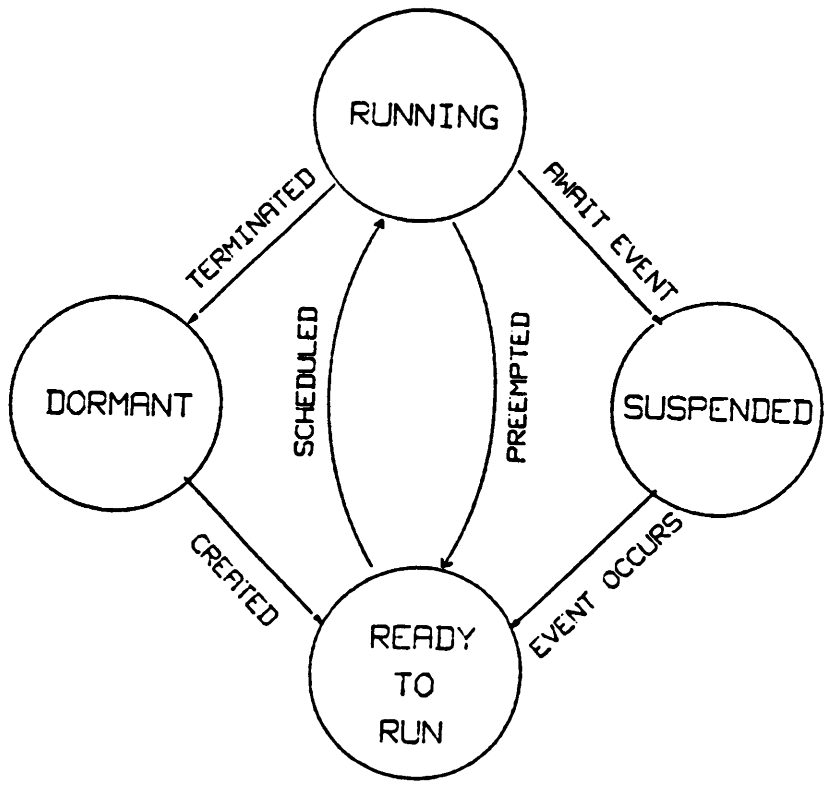


Figure 3. Task State Transition Diagram [23]

2.4 Scheduling

Scheduling is an integral part of multiprogramming systems as opposed to uniprocessing systems where only one process runs at a time. A set of policies and mechanisms are required and enforced to maintain the proper operations of computer system as a whole and processes running on it. Scheduler is an operating system module that controls the entry of new processes in the system and selects the new process to run. Scheduling policies are designed to maximize the throughput for particular type of system.

An operating system has many schedulers but there are three main type of schedulers that coexist and operate.
These are given below and are also shown in Figure 4 on page 17.

- **Long Term Scheduler** or job scheduler determines which jobs are to be admitted into the system for processing. It selects jobs from batch queue, which is normally reserved for resource intensive low priority programs. The primary purpose of this scheme is to provide the cpu with a balanced mix of compute-intensive and I/O-intensive programs.

- **Medium Term Scheduler** comes into play when a running process has to be swapped out because it goes in suspended state waiting for certain resources or data. It co-ordinates and maintains swapped out processes and tries to roll it back in when suspending condition is removed.

- **Short Term Scheduling** allocates the cpu among the ready to run processes resident in the main memory on the basis of predefined criteria. It is invoked whenever an event, internal or external, causes the global state of the system to change. Some of the most common events causing rescheduling are clock ticks, interrupts, I/O completions, system calls, activation of dormant tasks and sending and receiving of messages between tasks.

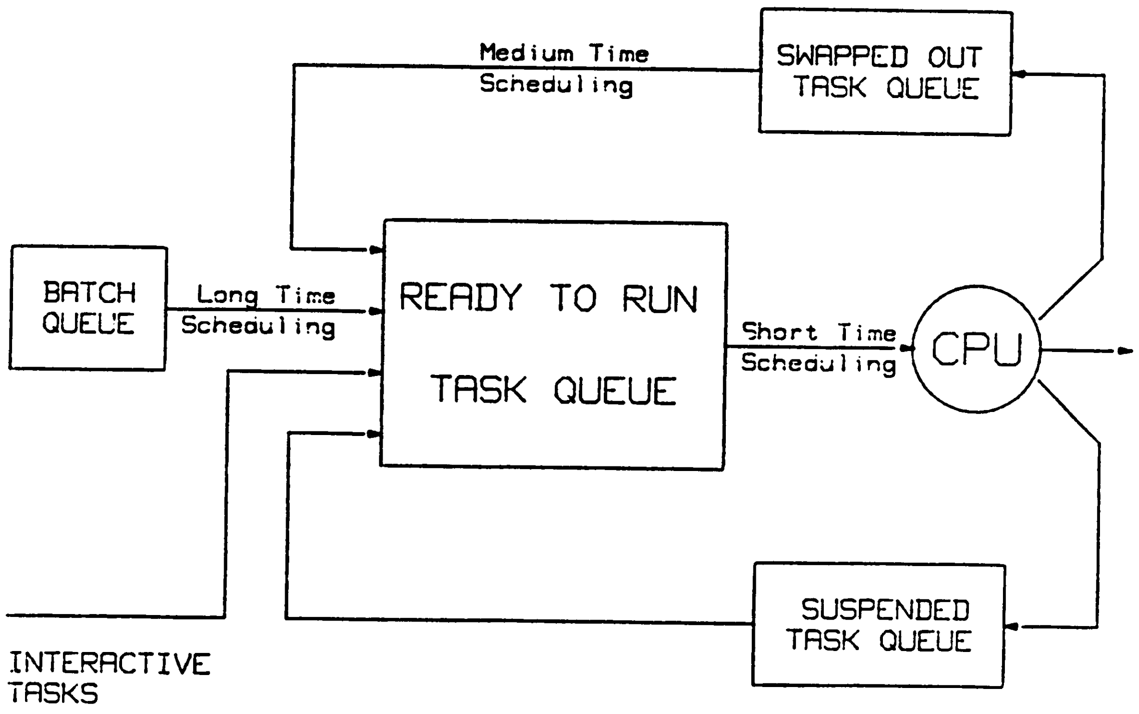


Figure 4. Three Levels of Scheduling [23]

2.5 *Scheduling Algorithms*

Scheduling algorithms are generally divided into two different classes namely preemptive and non-preemptive. In case of non-preemptive algorithms a task in control of the cpu will continue to execute unless it voluntarily relinquishes the control of cpu because it completes its operation and terminates or suspends itself waiting for a condition. In preemptive algorithms however any running process may be interrupted and replaced by a higher priority process. Preemptive schemes provide better responsiveness but entail greater overheads. These are heavily favored in case of real time systems.

2.5.1 First-Come-First Serve (FCFS) Scheduling

As the name suggests the jobs are executed in the order they arrive and preemption is not allowed. It can be easily implemented and overheads are very low. The performance however is poor and is totally inadequate for real time systems.

2.5.2 Shortest Remaining Time Next (SRTN) Scheduling

In this scheme the criteria for selecting the next process or job for execution is shortest remaining run time needed for its completion. It can be implemented with or without preemption. Non-preemptive version is called shortest job first (SJF) scheduling. It is optimal scheduling method for minimizing the average waiting time of a given work load [23].

2.5.3 Round Robin Scheduling

Round Robin or Time-Slice scheduling is essentially a preemptive algorithm since it operates on the principle of allocating cpu to each contending process for a fixed size time slice. This allows allocation of cpu time on a rotating priority basis and results in fair sharing of system resources among competing tasks. This method is very sensitive to the size of time slice thus making it most critical parameter. Very small time slice would entail increased number of context switches and increased overheads whereas larger quantum would degenerate into first-come-first-serve scheme.
This scheme is illustrated in Figure 5 on page 20.

2.5.4 Priority based Preemptive Scheduling

Each process in the system is assigned a priority on the basis of relative importance of the event it handles. The scheduler always selects for execution the task having the highest priority from among the ready to run tasks. The task priorities could be assigned statically or dynamically. A method called "aging priority", in which the priority of a waiting task is increased, is employed to ensure that low priority tasks are not locked out. Event driven scheduling is the preferred scheduling algorithm for real time multitasking systems since it provides good response time and reasonably balanced allocation of resources among different tasks.
This scheme is illustrated in Figure 6 on page 21.

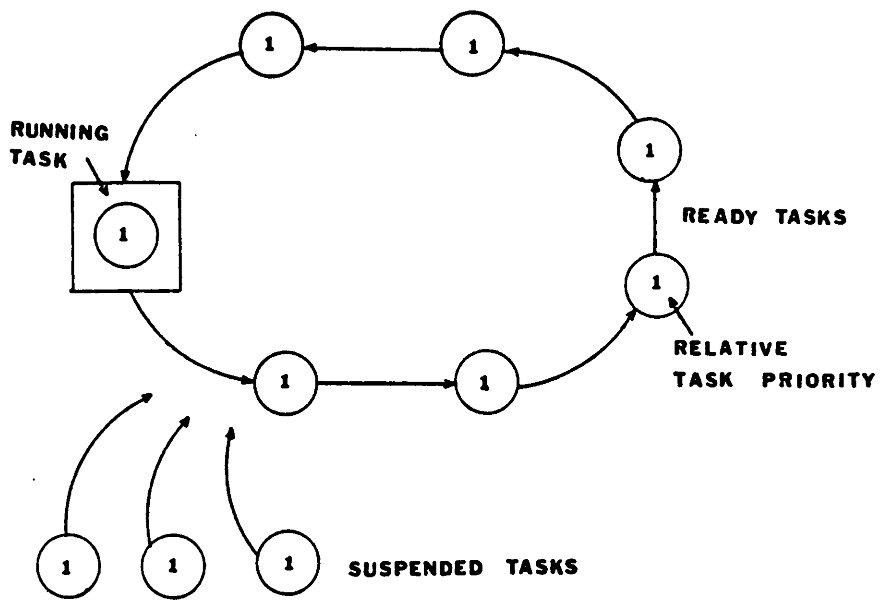


Figure 5. Round Robin Scheduling [21]

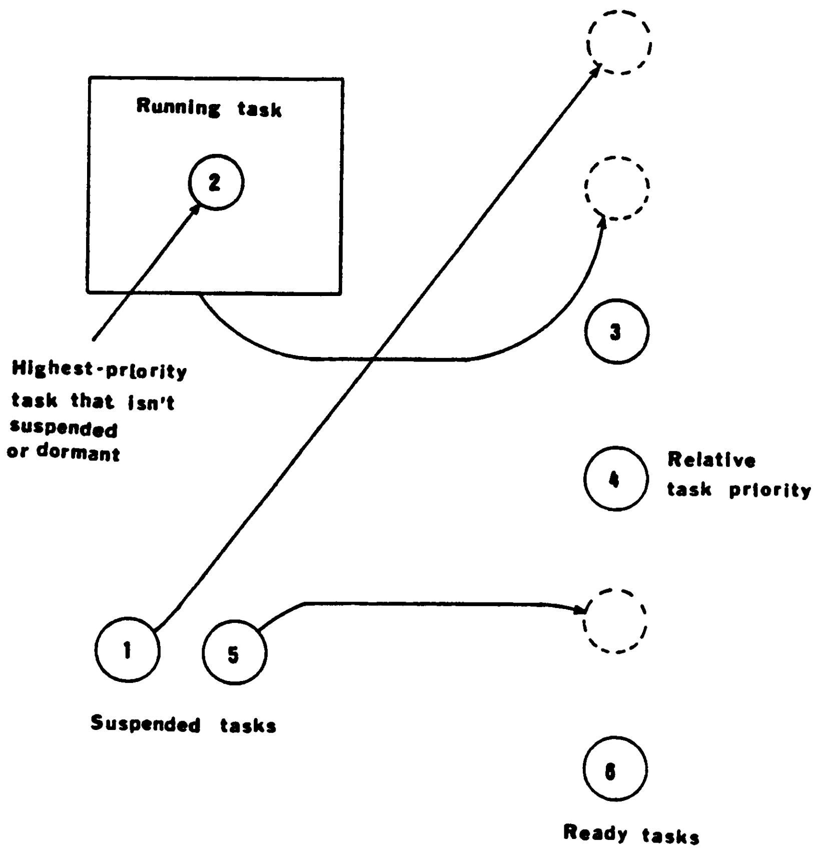


Figure 6. Priority-based Preemptive Scheduling [21]

2.6 Task Synchronization

In a multiprocessing or multitasking environments,there are usually many resources that are shared by different processors and or processes. Such shared resources include common memory and peripheral devices. One potential problem in such an environment is that of coherence which is particularly applicable in case of memory because a process would have modified certain locations and another process would use these new values in place of old ones leading to wrong results. Such occurrences should not be allowed to happen in a properly functioning system. Therefore the system must provide some mechanism to guarantee that asynchronous access to those resources is controlled in order to protect the integrity of the data.

Thus some form of mutual exclusion must be provided to enable one task to lock out access of a shared resource or variable by other processes or processors when it is operating on that variable or resource. This is called a critical section. A critical section is a code segment that once begun must complete execution before it, or another critical section that accesses the same segment or same resource can begin execution. Similarly I/O devices must be arbitrated among various tasks.

2.7 Synchronization Mechanisms

Inter-task communication is an essential and critical function in a multitasking environment. Inter process communication permits exchange of data between various processes which normally run asynchronously within a processor or on different processors. It is desired that cooperating processes must often communicate and synchronize [12]. In a typical application, one process may produce the data required by another process. The data from one process to another can normally be transferred through mailboxes, queues and buffers. Communication through buffers requires storing and retrieving data from it by different processes. These operations are indivisible and are controlled by providing mutual exclusion among the buffer operations. Synchronization is required in order to match the speeds of different processes and their respective rates of production and consumption [33].

Two types of synchronization are commonly employed when using shared variable. These are **mutual exclusion** and **condition synchronization**. Mutual exclusion ensures that a physical or virtual resource is held indivisibly. It might so happen that a shared data is in such a state that it can not be used for executing a given operation. Any processes which attempts to use this data for that operation should be delayed until another cooperating process modifies the data. It is sometimes called as condition synchronization.

One of the simple mechanisms to implement mutual exclusion among concurrent cooperating processes is realization of software locks. In this method either an instruction is provided (as TEST_AND_SET) or a small subroutine can be written (as in iAPX 8086) which can be used to enforce a synchronized access to shared variables. This scheme may result in performance degradation due to busy_wait or spin_lock phenomenon in which all the processes spend a lot of time in accessing and testing common variables.

Another approach defines two primitive operations called **block** and **wake-up**. Another synchronization primitive uses the semaphore, which consist of a counter, a process queue and two functions P and V. Event primitives are provided by two functions **wait** and **signal**. A process can wait on an event to become true and when another process signals an event then all processes waiting on the event are placed on ready queue. The messages provide a a flexible and direct method of interprocess communication by using primitives **send** and **receive** [12].

2.8 Semaphores

Semaphores are defined as variables which can be operated upon by primitives P and V to provide communication and synchronization between concurrent and cooperating processes. The primitives P and V are assumed indivisible and implement mutual exclusion mechanism. The semaphores are non negative integers which can be given a binary value when acting as a lock bit or they can take any integer values when they are used as resource counters [33].

The P procedure loops in a busy wait until semaphore is greater than zero,at which time it decrements it. The act of fetching, testing, decrementing and storing semaphore is considered an indivisible operation. The indivisibility of these operations is often ensured by some hardware mechanisms or instructions provided for this purpose. The V procedure increments the semaphore in a single indivisible operation. Given below is the description of P and V operations as it appears in the book by Hwang and Briggs [12].

The reader is referred to [1] and [3] for detailed discussion of concurrent programming and operating systems issues.

```
P(s) :   MUTEXBEGIN (s)

         s <- s - 1 ;
         If  s < 0 then
           begin
             Block the process executing P(s) and put it
             in a FIFO queue associated with semaphore s;
             Resume the highest priority ready to run process;
           end

         MUTEXEND

V(S) :   MUTEXBEGIN

         s <- s + 1 ;
         If  s <= 0 then
           begin
             If an inactive process associated with semaphore s
             exists,then wake up the highest priority blocked
             process and put it in a ready list.
           end

         MUTEXEND
```

3.0 Real Time Systems

The real time systems are characterized by a very distinctive feature that the results of some activity are to be produced within certain time frames dictated by the "real" world outside these systems. The duration of such time frames and frequency of such demands are the function of the process and the parameters being controlled. If the computations are not completed by the end of this interval, undesired consequences may result which are not particularly dependent on the amount of tardiness; a millisecond is as bad as as a second [20].

A computer system intended for real time applications provides a control function (f) that produces an output (z) based on occurrences or changes in input conditions (x). The designation "real time" stipulates that function (f) is continuous with time. This means that any change in input conditions causes the system to produce desired results within a specified length of time [42].
This relationship is shown in Figure 7 on page 27.

There are at least three measures of time which apply to real time systems: response time, survival time, and throughput or bandwidth [34].

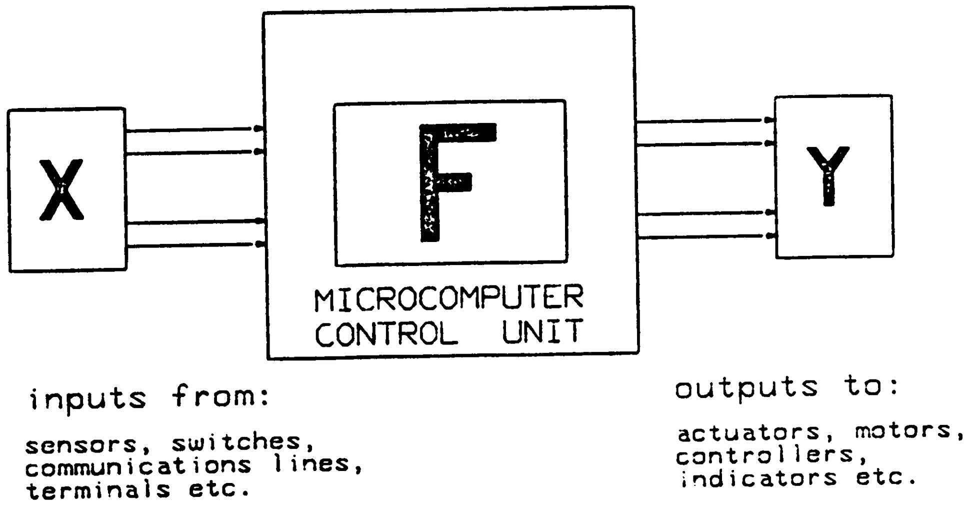

X
F
MICROCOMPUTER CONTROL UNIT
Y
inputs from:
sensors, switches, communications lines, terminals etc.
outputs to:
actuators, motors, controllers, indicators etc.

Figure 7. Block Diagram of a Real Time System

- **Response Time** is the amount of time elapsed before the computer is able to recognize and respond to an external event which caused change in its input conditions. This parameter is of critical importance in control application.

- **Survival Time** is the time within which the system has to accept or notice the information being made available to it. It is different from the response time in the sense that the data may be valid for a very short interval without requiring rapid response or it may stay long after any response would be futile.

- **Throughput** provides a measure of system performance in terms of its capability to handle number of events per unit of time. It may be reciprocal of average response time but is often less than that due to system overheads.

- **Bandwidth** is the same as throughput except that it is expressed as bits per second. It can be related to the analog notion of bandwidth. If sampling is involved it is important to distinguish between the bandwidth of analog signal and that of digital signals which is product of sampling rate and sample width in bits.

3.1 Applications

Today microcomputers of all flavors, sizes and capabilities are being used to implement real time systems spanning a wide range of applications. Most applications for real time systems are ***embedded***, in that the computer is built in a larger system and is not apparent to the users. Embedded microprocessors are to be distinguished from stand-alone type microcomputers such as small business systems or word processors.

A majority of embedded microprocessor applications fall into loose category of "process control". In such applications one or more microprocessors, buried inside a large system, are used to control some ongoing process in the outside world. A microprocessor is not apparent to users of automobiles or washing machines but it is likely to be there. More elaborate process control applications include numerically controlled (NC) machines, chemical processes, assembly lines etc.

Other applications involving embedded microcomputers are multiplexers, switches, telephone exchanges and modems in communications, intelligent analysis and measurement instruments, intelligent terminals and smart peripherals, robots and host of other industrial and consumer products.

3.2 Hardware Requirements

The hardware required for real time microcomputer systems is not much different from the hardware of other microcomputers. Normally it will consist of a CPU, memory and peripheral chips. There are also some hardware items rarely found in other than real time systems: analog-to-digital and digital-to-analog converters, relays, solenoids, stepping motors and so on [34]. These form part of the interface between the microcomputer and the real world. A block diagram of a typical real time system is shown in Figure 7 on page 27.

3.3 Software Requirements

The performance and operational requirements for the software that runs on the embedded systems are very different and stringent compared to software meant for stand-alone systems. The most important of these is **real time responsiveness** [38]. The system must be able to respond to random and unexpected events in the outside world quickly enough to be able to control the ongoing process.

In real time systems the function, represented in Figure 8 on page 31, is composed of many separate subfunctions and subprocesses called tasks. The system software must co-ordinate the execution of these tasks concurrently since events in the real world usually overlap and their occurrence is random. In addition to concurrent execution support, system resources must be managed so that a process does not interfere with the execution of other processes. This makes **multitasking** a key requirement for such systems.

A common set of mechanisms is necessary to support real time systems. In embedded applications these mechanisms are collectively referred to as real time operating systems or real time executives and serve as a foundation upon which the rest of application software is built. There are three basic aspects of the real-time executive:

- real-time program execution
- priority scheduling and reentrancy
- interrupt processing

Combining these three components into an integrated system results in an executive operating system with great capabilities.

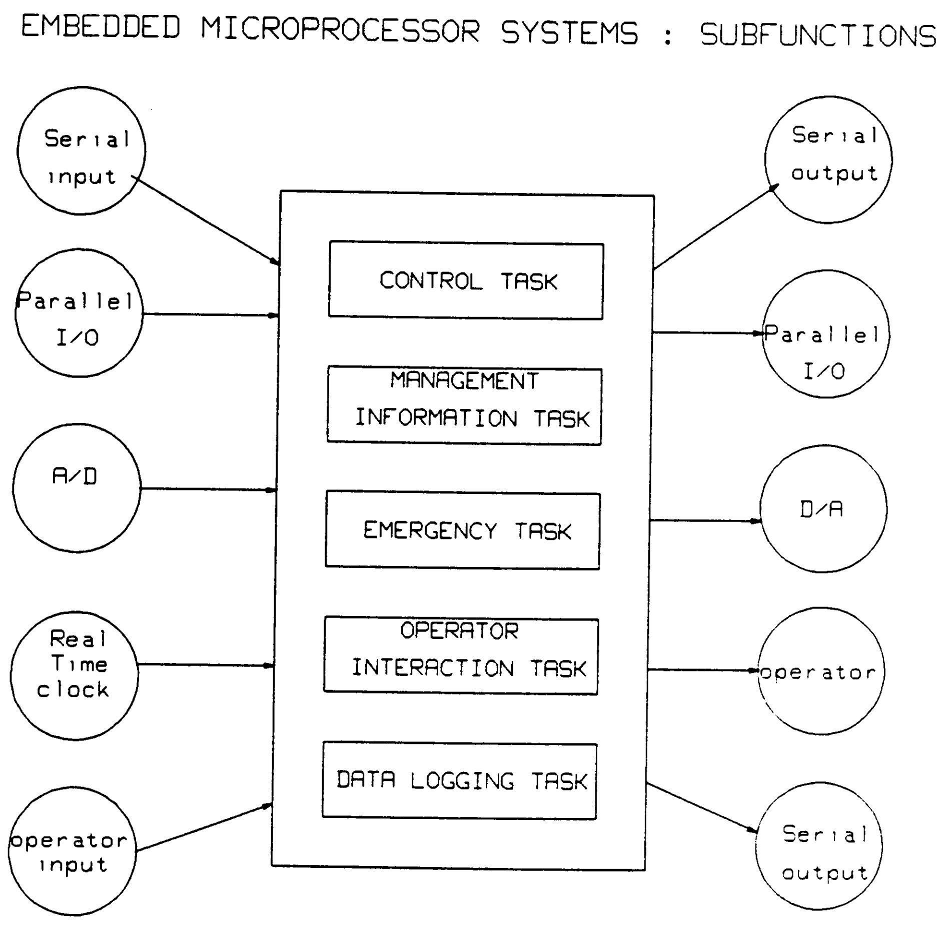


Figure 8. Subfunctions and Subprocesses [40]

A real time executive is a program, or operating system capable of handling a large number of tasks more or less simultaneously on a priority basis. These tasks may include program scheduling, interrupt servicing, peripheral communications and others.

The Figure 9 on page 33 depicts a conceptual representation of such a system.

3.4 Multitasking Systems

The fundamental concept behind multitasking system is overlapped task execution or in other words **concurrency**. Multitasking system designs are predicated on the fact that the "real world events" are slow on the average, compared to the speed of processor operations [42]. Multitasking can be described as a technique which allows the system to perform useful work between external events while still maintaining the ability to respond to external events in a timely manner as and when they occur.

This is achieved by switching the processor between several programs normally called tasks. The switching of the processor from one task to another task is called as **context switching**. It involves saving the current process state in sufficient detail so that this process can be re-started from the same point at which it was stopped and in the same environment at a later time when it gets processor attention. In real time systems context switching is usually caused by an external event that generates an interrupt to get the processor attention.

A Multitasking Operating System manages sharing of system resources among various tasks in the system so that system appears, externally, to be executing a number of tasks simultaneously. These resources include processor, memory, input/output and disk files. The part of multitasking system that manages the tasks and communication between them is called

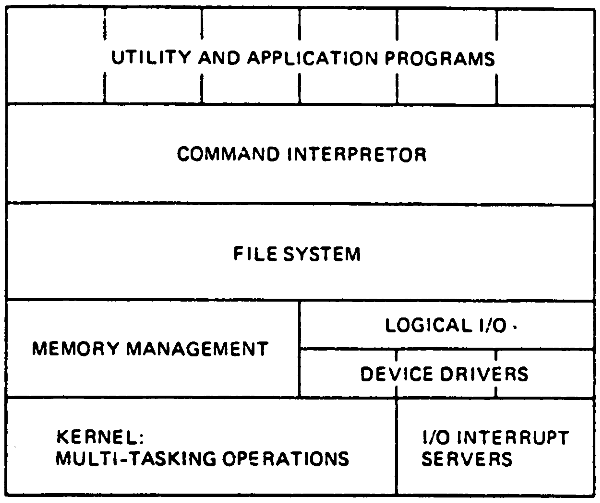


Figure 9. Conceptual Representation of a Real Time Executive [34]

kernel and is often augmented by routines to manage input/output operations. In a multitasking environment a given task may be started and stopped many times by the operating system before its completion. This intermittent execution, the result of system responding to external events, is totally transparent to the task itself. The only impact of this is the varying rate of execution.

Unlike foreground/background systems control is freely passed among the various tasks in a multitasking system. Multitasking is useful when the situation is quite complicated and can not be effectively dealt with by a single background task and interrupts [34]. The existence of many processes of approximately same urgency or priority at the same time would significantly degrade the performance of a uniprocessing and foreground/background systems. A multitasking system on the other hand would be able to perform quite satisfactorily in a similar scenario. Also see [2], [4] and [17].

3.5 An Overview of VRTX

Hunter and Ready manufacture and sell certain software resident in ROMs which is almost absolutely relocatable and independent of hardware environment. These are called as **Silicon Software Components**. They can be used in various configurations to achieve desired results in software much in the same way hardware devices are used to realize a particular function. The most important of these silicon software components are a Virtual Real Time Executive (VRTX), an Input Output Executive (IOX) and a File Management Executive (FMX).

The Figure 10 on page 35 shows a complete system incorporating all VRTX/OS components.

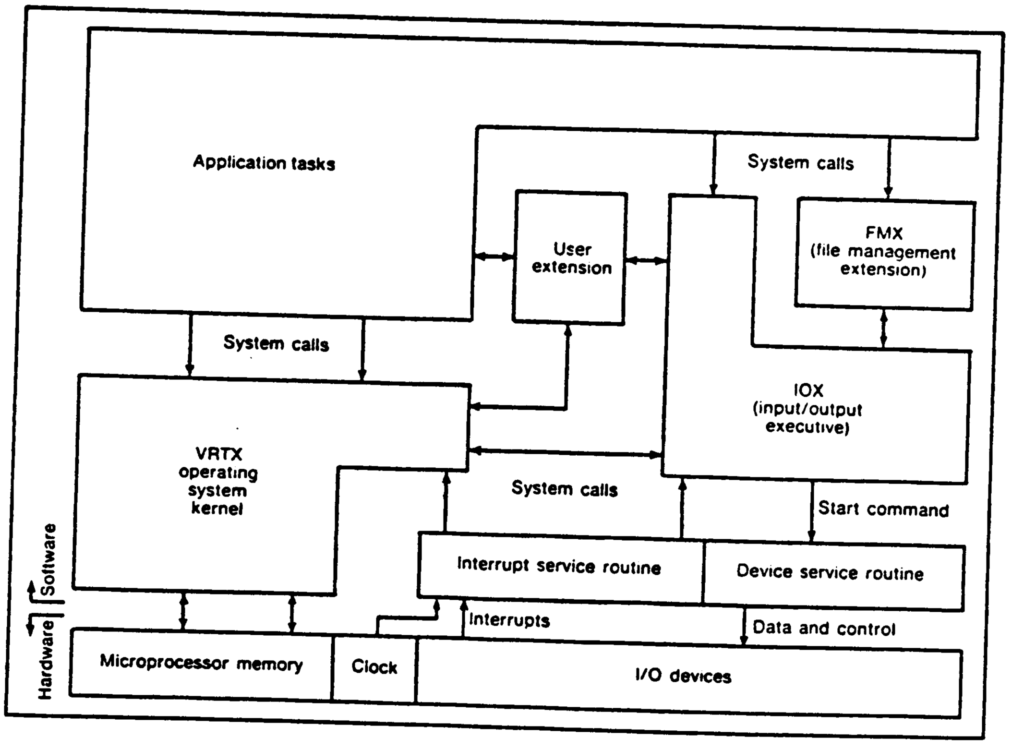


Figure 10. VRTX/OS architecture [38]

A silicon software component is an executable version of a microprocessor program that can operate on all board level microcomputers using the same microprocessor. Since code never needs to be modified it can be ROMed. These components are particularly helpful in embedded microprocessor applications such as intelligent terminal, robot, peripheral controller etc. They provide a foundation upon which the rest of the application software is built.

VRTX features can be summarized as below [38].

- Multitasking Support
- Interrupt-driven Priority-based Scheduling.
- Intertask Communication and Synchronization.
- Dynamic Memory Allocation.
- Real Time Clock Control with optional Time Slicing.
- Character I/O Support.
- Real Time Responsiveness.

Application software can easily be integrated with VRTX and can include user-defined system calls and device drivers. VRTX can be positioned at any paragraph boundary that is, at any physical address divisible by 16. Please also see [15] and [38] for further reading.

3.5.1 Configuration

User can specify all the parameters required by VRTX for a particular system environment in Configuration Table. This table and device specific interrupt handlers provide the interface between VRTX and its environment. One vector in Interrupt Vector Table (IVT) of iAPX86 points to VRTX entry location and a second vector points to the base of Configuration Table. These two pointers are the only direct link between VRTX and board environment. The default location of VRTX pointer is 80 Hex corresponding to INT 20H (INT 32) and that of Configuration Table pointer is 0200 Hex corresponding to INT 128.
The logical linkage of VRTX with target hardware is shown in Figure 11 on page 38.

Fields in Configuration Table describe system-managed memory, multitasking controls, interrupt support, linkage to other silicon software components and location of any user supplied routines for handling a special event. Other important entry in this table is that of a pointer to a character output routine called TXRDY. It is required if VRTX's character I/O support is used.

Please see the section on development of board support package for further details on various fields and parameters required for initialization.

3.5.2 Architecture

A system based on VRTX is layered according to function with each level making use of the functions provided by the level below. The hardware is the bottom most level and above it resides simplest, most hardware dependent operating system functions of software and on top are the user defined application programs. Each level defines a ***virtual machine*** for the level

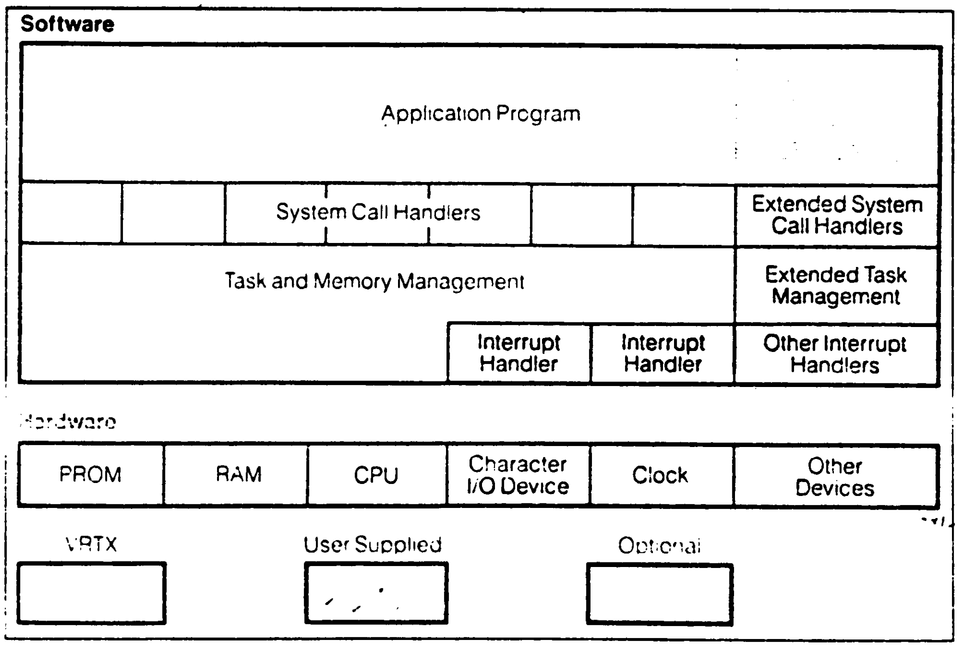


Figure 11. Logical linkage of VRTX and target hardware [30]

above it. The layered approach permits functions at various level to be concerned only with layers right above and below itself. This results in cleaner interface and efficient implementation.

The layered architecture of VRTX is shown in Figure 12 on page 40.

3.5.3 Task Creation and Management

VRTX is a multitasking operating system and its primary responsibility is management of various tasks active in a system. The task is smallest set of activities constituting some meaningful function. This is the logical unit which is created, managed and deleted by VRTX freeing the user from chores of management. It is capable of supporting 256 logically distinct and active tasks which does not include dormant tasks.

VRTX creates a task when it receives a **sc_tcreate** from the application program which provides the code for this task. Every task is assigned a unique identification number as well as a priority level at the time it is created. Many tasks may be assigned to same priority level. The priority levels take on values ranging from 0 to 255 with 0 being the highest level. The priority levels are used by the VRTX to schedule the tasks when they become ready to run.

Normally there is one main task in the system which creates the remaining tasks. This task may delete itself in certain cases after creating other tasks. It is the first task to be created and is created before multitasking starts. Once multitasking comes into action the tasks are given control of the processor and other resources in accordance with their priorities. At any given point of time the task with the highest priority and with no pending condition is the one which gets to execute. Any task can preempt another task if the running task has lower priority.

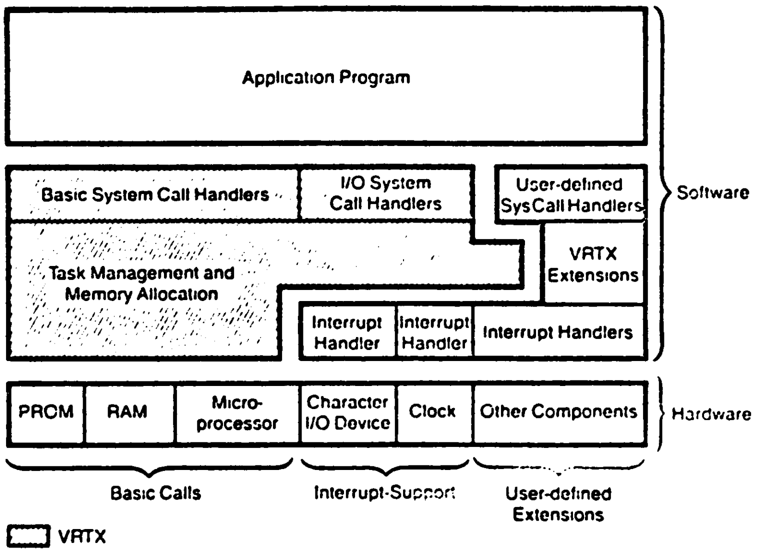


Figure 12. VRTX Architecture [38]

3.5.4 Intertask Communication and Synchronization

VRTX provides a set of very elegant mechanisms to accomplish intertask synchronization and communication. It is fairly obvious that in any given multitasking environment various task would constitute a process or application, that is, tasks are created by dividing some large function into smaller units. If these tasks have to produce the desired response they have to cooperate to a certain extent. This entails a need for communication between various tasks as well as synchronization.

VRTX provides two data structures namely **mailboxes** and **queues** to realize communication and synchronization among tasks. These two data structures are to be defined in the application program so that necessary storage is allocated. The system provides three specific system calls to access these mailboxes and queues. These are

sc_post	post a double byte message.,
sc_pend	pend for receiving the message.
sc_accept	retrieve the message from the mailbox

Whenever a task has to communicate with another task it will do so by posting the message on the mailbox on which the recipients task will pend. It is evident that this simple mechanism can be effectively used to achieve mutual exclusion. By assigning timeout value to a pending task it is also possible to let it go after it has waited long enough for a message which never came. This can avoid possible deadlock situation as well as induce fairness. The sc_accept call is used to retrieve the message from a mailbox. If there are more than one task pending at a particular mailbox then the highest priority task among them will receive first any arriving message. For more details see [6] and [27].

Another important data structure supported by VRTX is a queue which is extensively used for queuing up the messages making them a prime choice for synchronization. These can be used to implement counting semaphores and SIGNAL and WAIT constructs. The following system calls are used to manipulate the queues.

sc_qcreate create a message queue.

sc_qpost post a message to queue.

sc_qpend pend for a message from the queue.

sc_qaccept accept a message from the queue.

sc_qinquiry queue status inquiry.

3.5.5 Interrupt Support

Interrupts are part and parcel of real time systems as they signal to the system various events that are occurring asynchronously outside its environment. A good interrupt response capability and support thus becomes a very critical feature. VRTX allows the existence of user supplied event or device specific interrupt handlers which are given control of the processor in case of an external event. Thus the multitasking activity ceases for the period an interrupt is being serviced. They also serve as vital inputs or message carriers for many tasks in the system which depend on external events to get to run.

VRTX permits certain system calls to be issued from within the code of interrupt handlers. The most widely used calls among them relate to posting and accepting messages from mailboxes or queues.

4.0 Porting VRTX/88

The only assumption VRTX makes about the hardware environment in which it runs is the presence of a VRTX supported microprocessor [11], which is an 80C88 in this case. VRTX/86 provides a true chip level support for entire iAPX/86 family processors and requires only the CPU and a small amount of memory. Hence it can be used without any modifications on variety of boards containing these microprocessors.

However VRTX must be interfaced with the target hardware system to be able to provide various multitasking operating system services. This is achieved by providing the relevant information about the hardware environment in a predefined fashion expected by VRTX. This information is used by VRTX to establish logical connection with certain hardware components in the system.
It is shown schematically in Figure 13 on page 44 as to how the board support package provides such connections [11]. A board support package thus forms an interface between VRTX and the particular microprocessor board it runs on.

A board support package is a collection of a number of related items as discussed below.

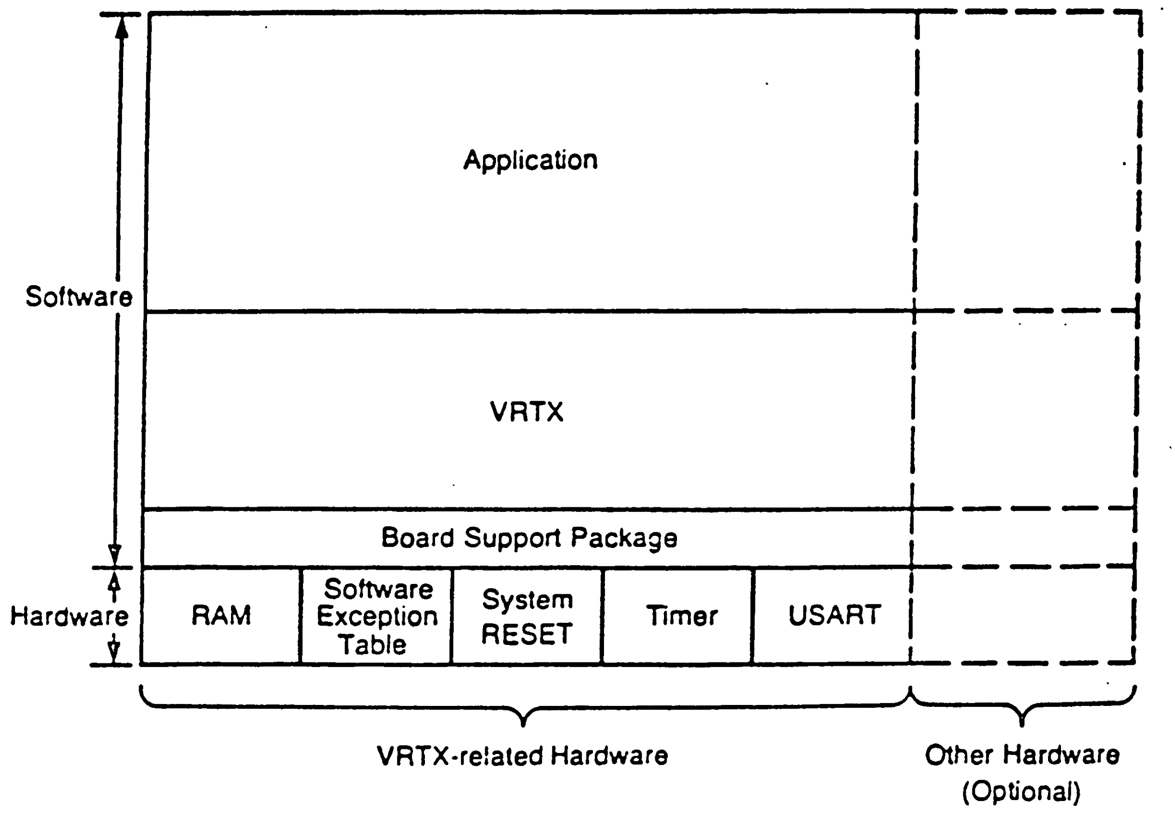


Figure 13. Board Support Package [11]

4.1 Environmental Data

This information pertains to the details of the particular hardware on which VRTX is to be installed.

1. **Key Addresses**, which define various I/O registers, memory locations containing read write memory (RAMs) and read only memory (ROMs), the base address of VRTX PROM and its entry point etc. These points have been discussed in detail in following sections.

2. **VRTX Configuration Table**, which is the collection of various address pointers and size data. The pointers point to different locations for use by VRTX and size data defines its workspace. Other important parameters like stack size, separate interrupt stack and user supplied extensions are also specified here.

3. **VRTX Entry Pointer** is a vector loaded into the interrupt vector table (IVT) of the processor and it points to the entry point in VRTX. The default for this vector is INT 32 but another vector can be specified.

4. **VRTX Configuration Table Pointer** is another interrupt vector, which is to be loaded into IVT of processor by the user. It points to the Configuration Table which is a prespecified data structure used for supplying environmental information to VRTX.

4.1.1 Initialization Code

This refers to **user defined initialization**, which loads these pointers into Configuration Table, initializes various I/O and peripheral devices like USARTs, Timers etc. in the system and invokes **VRTX Initialization**, which creates VRTX's system variables and sets up user defined stacks.

4.1.2 Interrupt Service Routines

In order to use timing and character input/output services of VRTX two interrupt handler must be provided. These are **Timer Interrupt Handler**, which translates a timer interrupt into a VRTX UI_TIMER call and **USART Interrupt Handler(s)**, which translates USART interrupts into UI_TXRDY and UI_RXCHR system calls.

4.2 Developing a Board Support Package

This section describes various steps to be followed in designing a board support package (BSP) for a microcomputer board. The procedure described here was used to develop a board support package for Microsystems International's MSI-C988 microcomputer board. The details of the hardware on this board are given in Appendix-B. The procedure is fairly straight forward and may require only minor changes for different hardware configurations.
Please see Figure 14 on page 47 for the outline of board support package development process.

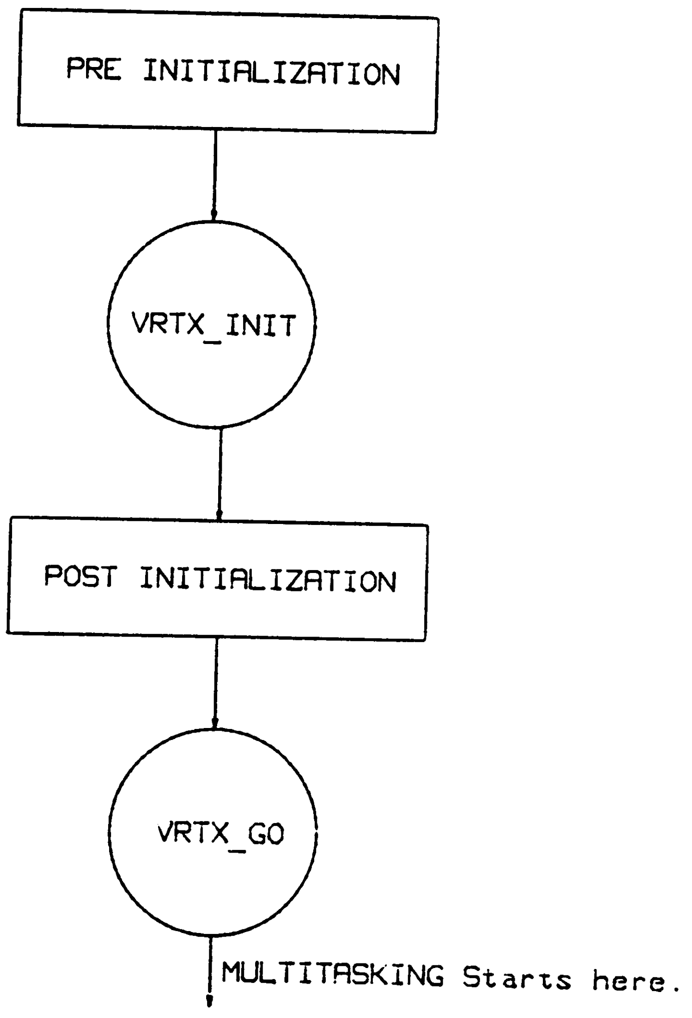


Figure 14. General Structure for Board Support Package Design

4.2.1 Board Architecture

It is very important to know the architecture of the target microcomputer and the interface it presents to the software. Notice that a microcomputer board can be designed in many ways using variety of devices. The first step is to study and understand the specifications and operation of the target board with special attention to input output devices and their interfacing and interrupt handling. This can be achieved in steps given below.

4.2.1.1 Memory Map

Document the **Memory Map** specifying the addresses and the layout of the memory available in the system. It would be wise to incorporate in the map the memory which can be added or will be added. Note that memory map pertains to entire microcomputer system which will run VRTX. The next step is to designate, if you are configuring the boards, or mark different areas used by RAM or PROM devices. Also note any empty slots in entire address space. A well documented memory map will

- account for entire feasible memory
- identify each block as RAM or ROM
- indicate the slots occupied by monitor if any and
- mark the holes in the address space that is unused blocks in a system.

The memory map for the target microcomputer used in this case is shown in Figure 15 on page 49.

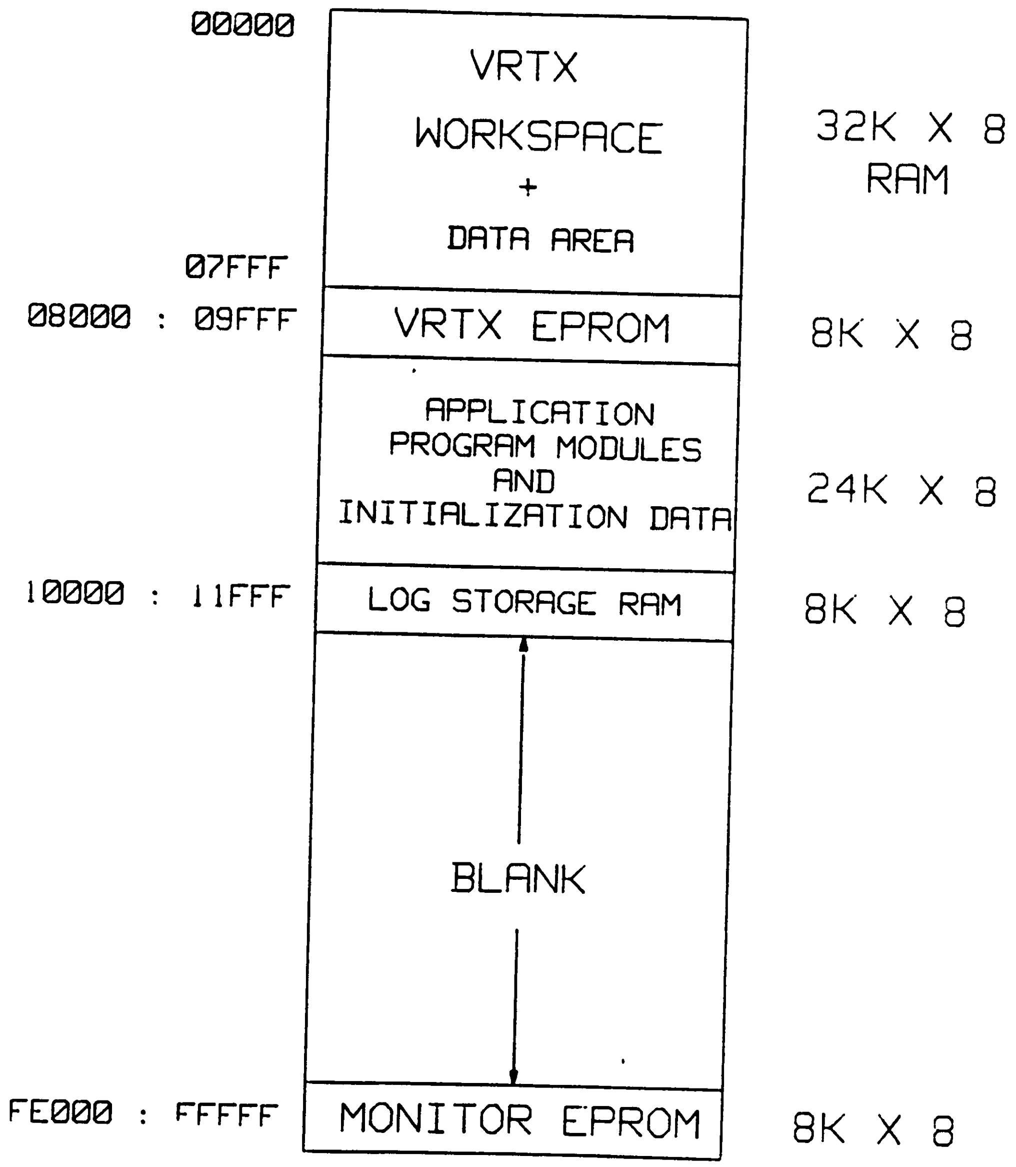


Figure 15. Memory Map for MSI-C988 Microcomputer and MSI-C764 Memory Board

4.2.1.2 I/O Map

More or less similar procedure is to be followed in documenting the Input Output address of the microcomputer. It is different from the memory map in the sense that we need to be concerned only with the devices already installed or likely to be added. The documentation of board can be used to find out the device addresses for all of I/O devices in the system. It entails locating the addresses of various registers of devices like USARTs etc.
The I/O address map for the system under consideration is shown in Figure 16 on page 51.

4.2.1.3 Device Programming

In order to be able to program various I/O devices in the system in the way the application requires we need to know the operation of these devices as well as the steps involved in programming them. The best place to find such information is the data sheets supplied by manufacturers of those devices. However they are likely to be found in the documentation of the board on which they are used.

The programming of I/O devices for a particular mode of operations will be required to setup the board for any meaningful purpose. It is clearly evident from the pseudocode of board support package presented later in this chapter.

4.2.1.4 Interrupt Structure and Control

The interrupt structure though in no way dictated by VRTX, is very vital to performance of the system. The board support package must initialize the interrupt controller(s) in the system in

I/O Device	Register	Address
USART 1	Usart Control Register	01H
82C52	Modem Control Register	02H
	Baud Select Register	03H
	Data Register	00H
USART 2	Usart Control Register	21H
82C52	Modem Control Register	22H
	Baud Select Register	23H
	Data Register	20H
PIT	Control Register	63H
82C54	Counter 0	60H
	Counter 1	61H
	Counter 2	62H
PIC	Control Port	40H
82C59A	Data Port	41H
RTC	Counter(1/10000)	C0H
	Counter(1/10,1/100)	C1H
	Counter(sec)	C2H
	Counter(min)	C3H
	Counter(hour)	C4H
	Counter(day of wk)	C5H
	Counter(day of mo)	C6H
	RAM(1/10000)	C7H
	RAM(1/10,1/100)	C8H
	RAM(sec)	C9H
	RAM(min)	CAH
	RAM(hour)	CBH
	RAM(day of wk)	CCH
	RAM(day of mo)	CDH
	Interrupt Status	D0H
	Interrupt Control	D1H
	Reset Counters	D2H
	Reset RAM	D3H
	Status Bit	D4H
	GO Command	D5H
	Standby Interrupt	D6H
	Test Mode	DFH

Figure 16. I/O Map for MSI-C988 Microcomputer System.

conformance with the overall interrupt structure. The type of Interrupt Controller and its selected mode of operation will have system-wide ramifications [11]. This also affects the design of interrupt service routines. For example if Auto-End-of-Interrupt feature of 8259A PIC is not used then the service routine must send an EOI command.

The interrupt structure of our target microcomputer board is depicted in Figure 17 on page 53. In this case each USART is made to generate both Receive and Transmit interrupts via interrupt controller. Similarly the Programmable Interval Timer (82C54) outputs or Real Time Clock (55167) outputs can be used as interrupts.

4.2.2 System Environmental Data

The VRTX is to be provided with the necessary data about system environment so that it can implement a logical interface with the hardware. The basic purpose of a board support package is to pass this information to VRTX in a predefined data structure. A board support package typically begins with the documentation of the memory and I/O maps, followed by the Configuration Table [11].

The following addresses are to be supplied to VRTX by a BSP. They are,

- The **Base Address of VRTX** that is where the PROM containing VRTX/88 is installed. In case of VRTX/86 it refers to location of lower-addressed PROM (labeled PROM - A) which must be aligned at a paragraph boundary. In our case the VRTX/88 PROM has been installed at physical address of 0A000 Hex.

- **VRTX's system call entry point** which is same as its base address in this case but may be different in case of multiple silicon software components.

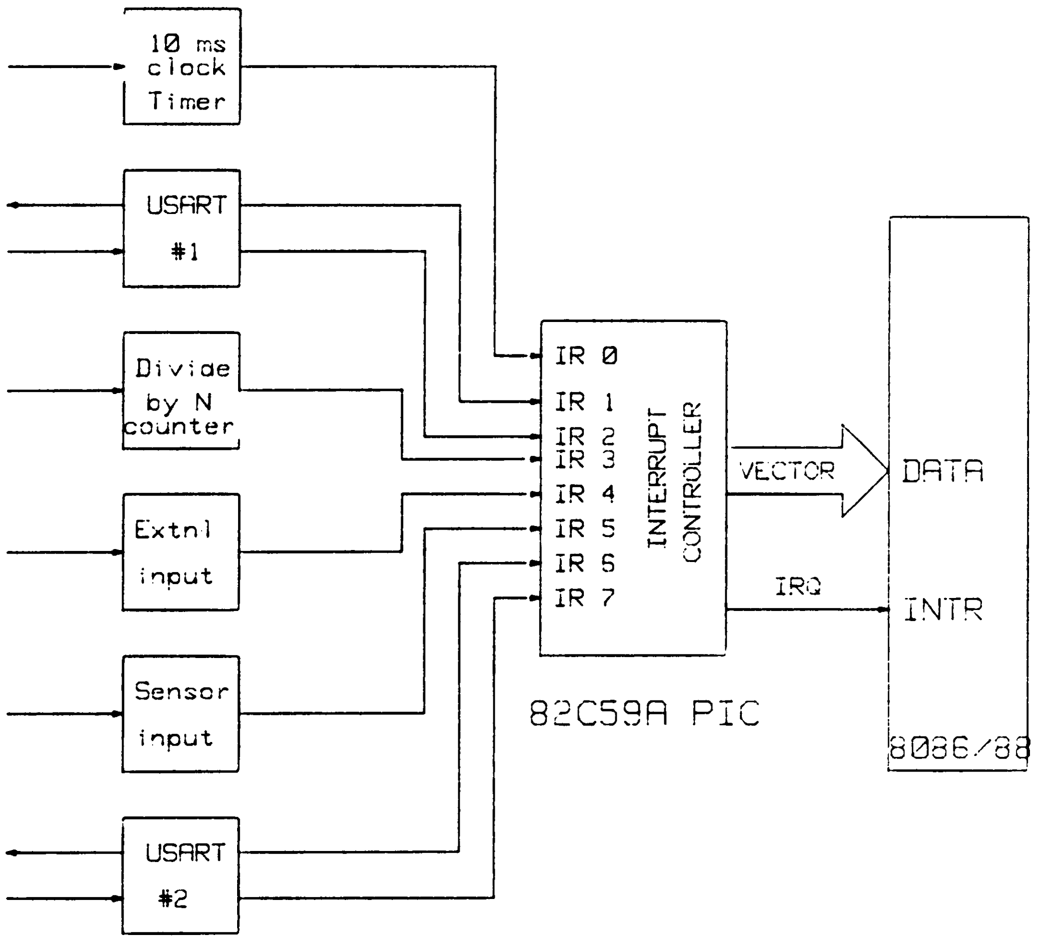


Figure 17. Interrupt Structure for MSI-C988 Microcomputer Board.

- The I/O device register addresses.

- The Interrupt Vector Table (IVT) for the target microprocessor which in our case is from 00000 to 003FF Hex the reserved area for 8088/8086 IVT.

4.2.3 Configuration Table

The VRTX has a predefined data structure called Configuration Table to ensure orderly and consistent way of passing system environmental data to it. It is a 48 byte data structure with 16 different fields specifying different system memory parameters, task parameters and special hooks.
The diagram in Figure 18 on page 55 depicts the structure and the fields of the Configuration Table. At initialization this table supplies VRTX with following information.

- Location and size of VRTX workspace.

- Number of tasks in the system.

- Stack sizes for interrupts and tasks.

- Addresses of user extensions.

- Location of Component Vector Table.

The **VRTX-workspace-addr** field contains a pointer to the area which has been reserved by the user for VRTX's own use. VRTX maintains a 256 byte area for system variables, a Task Control Block (TCB), stack for each active task in the system, interrupt stack and all control structures for queues and partitions in this workspace. The pointer is specified as a segment address signifying 16 most significant bits of 20 bit physical address [38].

	VRTX/86
0	VRTX-workspace-addr
2	VRTX-workspace-size
4	(reserved, must = 0)
6	ISR-stack-size *
8	(reserved, must = 0)
0A	(reserved, must = 0)
0C	(reserved, must = 0)
0E	user-stack-size
10	(reserved, must = 0)
14	user-task-count
16	(reserved, must = 0)
18	TXRDY-driver-addr
1C	*sys-TCREATE-addr
20	*sys-TDELETE-addr
24	*sys-TSWITCH-addr
28	*CVT-addr

Figure 18. VRTX Configuration Table [38]

The **VRTX-workspace-size** field indicates the size of VRTX workspace in paragraphs (16 byte blocks). The required workspace size is to be calculated for each application. Please see next section for calculations. The size of VRTX-workspace, excluding user task stacks, can not exceed 4K paragraphs (64K Bytes).

The **ISR-stack-size** field indicates the size of a separate interrupt stack to run interrupt service routines (ISRs). This is 0 if ISRs are to use stack of interrupted task. It is to be specified as count of paragraphs if all ISRs are to use a single stack other than task stacks. The ISR stack if specified stays in VRTX workspace.

The **user-stack-size** field specifies the size of the stacks as count of 16 Byte paragraphs, to be allocated to tasks. The required stack size is largely a function of the task characteristics hence this field is allowed to be bypassed at initialization if it is 0. Now the user must explicitly allocate stack size while creating the task.

The **user-task-count** specifies the maximum number of tasks that can be simultaneously active in the system. It is different than the total number of tasks required for the application.

The **TXRDY-driver-addr** points to the TXRDY routine which will be used by VRTX for character I/O.

The sys-TCREATE-addr, sys-TDELETE-addr and sys-TSWITCH-addr are optional parameters which may have to be specified for special cases. Another optional parameter CVT-addr is to be specified as pointer to Component Vector Table (CVT) for routing to components other than VRTX like TRACER or IOX. Please see VRTX User's Guide for further details.

All parameters declared as **reserved** should be made 0 for proper operation. These are provided for future upgrades [38].

4.2.4 Determining Configuration Table Parameters

In this section the method of determining the Configuration Table will be illustrated using the example of MSI-C988 board. The first step is to find the required size of VRTX workspace. The VRTX User's Guide gives a formula that can be used to determine this size. It is,

$$\text{VRTX-workspace-size} =$$
$$[256 + 48*t + 10*p + 6*b + 12*q + 4*qe + s] / 16 + (us*t) + is + 1$$

where

t	Maximum number of tasks in the system.
us	User stack size counted in paragraphs (16 bytes).
p	Memory partitions in the system (10 bytes/partition).
b	Sum of partition size divided by partition block size for each partition.
q	Number of queues in the system (12 bytes/queue).
qe	Queue elements in the system (sum of all queue sizes)
s	if us=0 then s=64 else s=0.
Is	Interrupt stack size.

A sample calculation for a Board Support Package test program is given here. The parameters are, Number of tasks = 6

User stack size = 16 paragraphs

No Interrupt stack

No queues

No partitions.

Using the formula given above the VRTX workspace size can be calculated as,

= [256 + 48t] / 16 + us*t + 1

= [256+48*6]/16 +16*6+1 = 544/16 + 97 = 135 paragraphs

= 2160 bytes.

4.3 Initialization

The system initialization is performed in three phases: before the VRTX_INIT call, during the VRTX_INIT call and after the VRTX_INIT call [11]. The user is responsible for supplying the appropriate initialization routines for specific devices in pre-initialization period and post-initialization period. Rest of the initialization is performed by VRTX_INIT itself. These phases are discussed below.

4.3.1 Pre-Initialization

During this phase of initialization a vector is loaded in IVT which points to user supplied Configuration Table thereby providing a way for connecting VRTX to it. VRTX assumes, by default, that this pointer is located at location 0200 Hex corresponding to INT 128 but user can supply another vector if required. If another vector is chosen then an internal pointer inside VRTX must be changed to point to this one. This pointer resides at offset of 022H from VRTX entry point. Another parameter to be supplied at this time is VRTX entry pointer. Note that none of VRTX system calls can be used at this time as VRTX is as yet uninitialized.

4.3.2 VRTX Initialization

The VRTX initializes itself when it receives a VRTX_INIT call which is normally specified after pre-initialization phase. The user program following this call can check the value returned by VRTX_INIT which is ER_INI in case of error, to ensure proper initialization.

4.3.3 Post-Initialization

During this phase various I/O devices are initialized by invoking user supplied initialization routines.

A general structure of typical post-initialization code is shown in Figure 19 on page 60. Apart from device initialization another important activity during this phase is creation of the initial task.

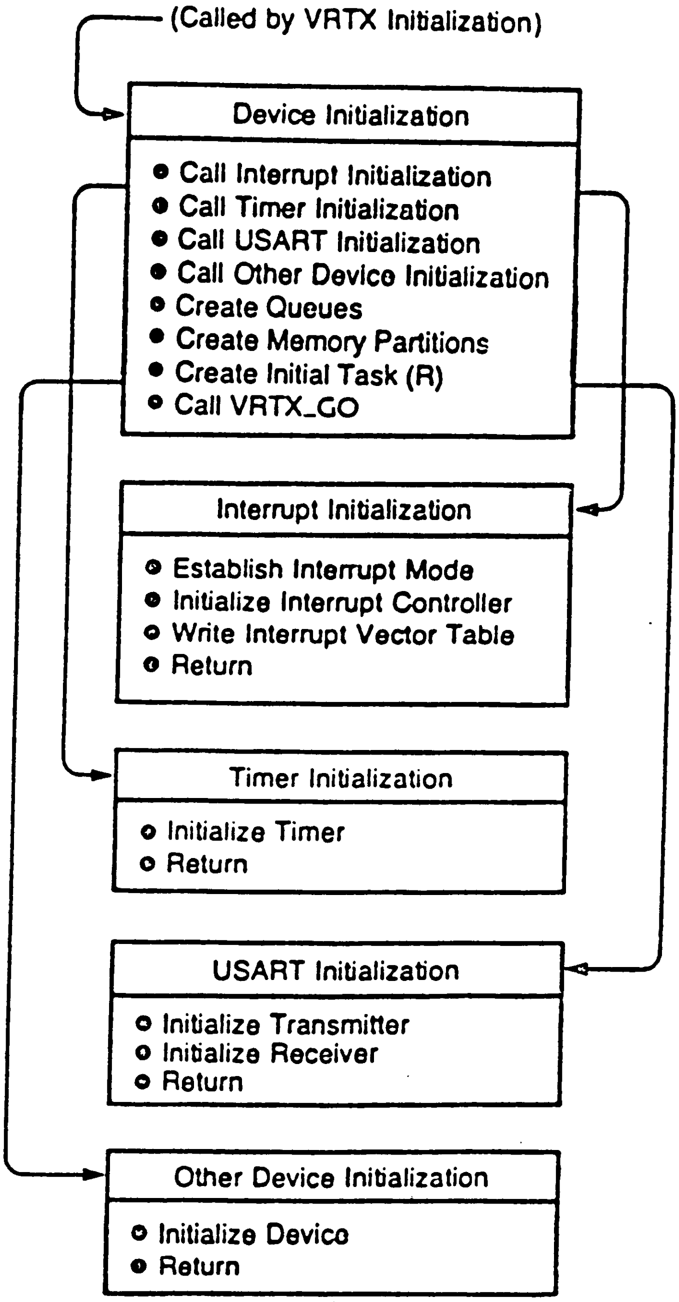

(Called by VRTX Initialization)
Device Initialization
• Call Interrupt Initialization
• Call Timer Initialization
• Call USART Initialization
• Call Other Device Initialization
• Create Queues
• Create Memory Partitions
• Create Initial Task (R)
• Call VRTX_GO
Interrupt Initialization
• Establish Interrupt Mode
• Initialize Interrupt Controller
• Write Interrupt Vector Table
• Return
Timer Initialization
• Initialize Timer
• Return
USART Initialization
• Initialize Transmitter
• Initialize Receiver
• Return
Other Device Initialization
• Initialize Device
• Return

Figure 19. General Structure of Post-Initialization Routine [11]

```
/* pseudocode for board support package */

          set up the Stack Pointer
          initialize
               {
               configuration table pointer
               VRTX entry pointer
               }
          call  VRTX_INIT
          initialize
               {
               Serial I/O (82C52)
               Interrupt Controller (82C59)
               Counter/Timer (82C54)
               Real-Time Clock (58167)
               }
          call VRTX_GO
```

Figure 20. Pseudocode for Board Support Package.

In case of the application considered here device initialization common to all devices was performed first and then subroutines specific to various I/O devices were called. Please see Figure 20 on page 61 for details. The initialization is designed in such a way so that it does not interfere with the onboard monitor program (C988-M). It was decided to retain the monitor to allow downloading from an IBM PC used for software development. The rules mentioned in BSP manual were adhered to which state that,

- Do not overwrite any Interrupt vectors used by monitor.
- Refrain from initializing any device already initialized by monitor or initialize them in the same way as monitor does.

4.3.4 USART Initialization

It is compulsory to initialize at least one USART in the system if the application program is going to use the character input output services provided by VRTX. Initialization of USART includes specifying baud rate, number of stop bits, number of bits per character, parity, handshake signals and modem control signals conforming to serial device interfaced to the system. An address pointer to the transmit driver needs to be specified in the configuration table to inform VRTX that character I/O is to be performed using this driver.

Except in certain specialized cases it is expected that USARTs will be operating in interrupt driven mode. It is convenient to have both transmitter empty and receiver full conditions in USART generate separate interrupts for good response but that is not mandatory. The application program uses VRTX supplied system calls UI_TXRDY and UI_RXCHR in interrupt service routines to output or input the characters.

4.3.5 Timer Initialization

The presence of a free running clock is often desirable in a real time system to sequence and delay activities. It can also be used for time stamping certain events for later analysis. VRTX does not need a clock to perform its basic functions however a free running clock is required if the TIMER services of VRTX are to be used by the application. The VRTX maintains a 32 bit timer which requires that a UI_TIMER call is made from an Interrupt Service Routine to signal one clock tick. The clock tick could be variable depending on the resolution required. Note that clock should be free running that is it should need not be loaded after every tick, and if it does then user supplied service routine should do it. It introduces certain amount of irregularity in the timing.

4.3.6 Interrupts

VRTX requires that all interrupt service routines (ISRs) must begin with UI_ENTER and terminate with UI_EXIT calls. The AX register must be saved on stack before issuing UI_ENTER call since it is used to make the call, it is restored by UI_EXIT call while returning.
The layout and structure of various pointers has been given in Figure 21 on page 64. The actual code is provided in the Appendix-G.

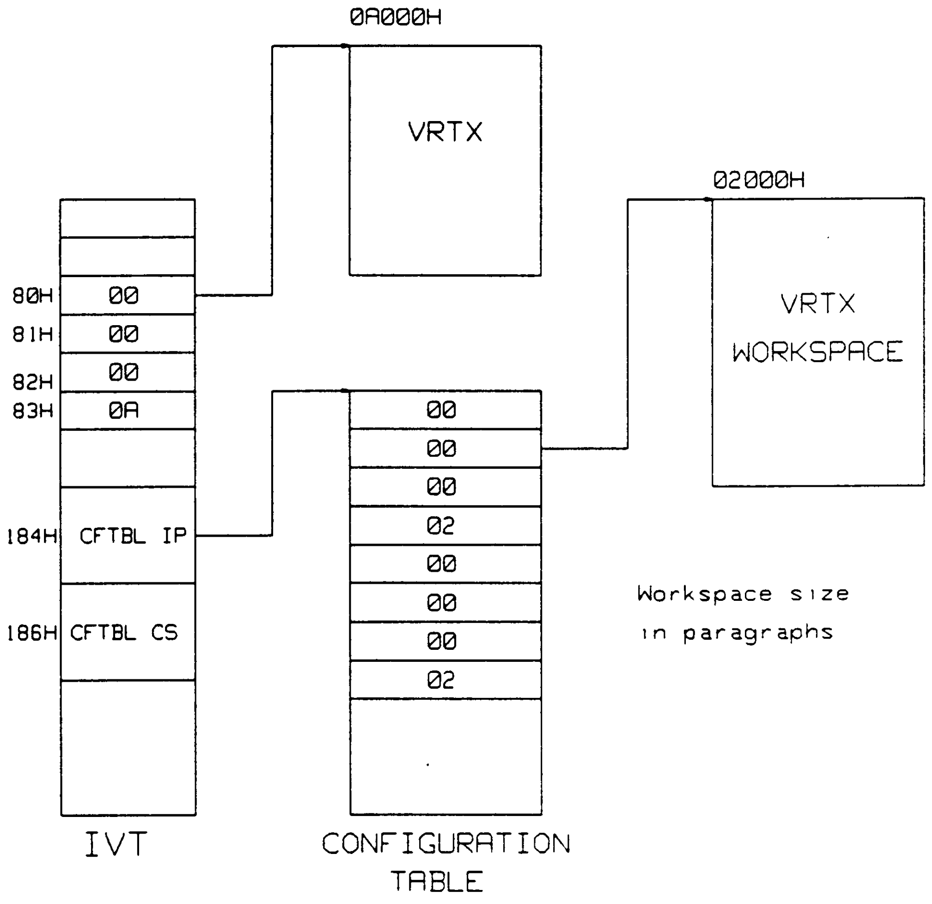


Figure 21. VRTX Initialization

4.4 Board Support Package for MSI-C988 Card

The previous sections of this chapter describe in detail the ground work required prior to actual design of a board support package (BSP). By following the step by step procedure outlined in previous section we have relevant data about the memory and I/O address space of the system, device type and programming information and details of interrupt control structures. The monitor imposed constraints are also known at this time.

The next step therefore is to make certain design decisions which should reflect the needs of the application. For the purpose of BSP however it is sufficient to know the input output and memory requirements of various tasks, number of tasks and some idea of task interface. The discussion about the importance and the values of these parameters is presented in detail in next chapter which deals with the system design aspects of the representative application. Most of these decisions directly affect the composition of Configuration Table.

The board support package developed for MSI-C988 microcomputer incorporated the following. The reader is referred to Appendix-G for actual code for this board support package and the test programs to verify its correct operation.

1. Pre-Initialization

   - Load in the double word configuration table pointer (cs:ip) at the location 0184H, that is, the offset is stored at 0184H and the segment at 0186H. Note that the default value for this pointer is 0200H.

   - Load in the VRTX entry pointer (cs:ip) at the location 020H corresponding to INT 32 (INT 20H).

2. VRTX Initialization

- Issue VRTX_INIT call (function code 030H) to allow VRTX to initialize its parameters in anticipation of forthcoming system calls from post-initialization phase and subsequent multitasking.

- Check the return code for successful initialization and take corrective action if not.

3. Post-Initialization

- Program the programmable interrupt controller 82C59A in buffered master mode with interrupt vectors starting at 020H (INT 8). Other initialization parameters are edge triggered interrupts, 8086 mode, interrupts IR0 through IR7 enabled and normal End-of-Interrupt cycle.

- Program the first 82C52 USART with following parameters.

  - 8 bits per character, 1 stop bit, no parity

  - 4800 baud, DTR = 0, CTS = 0, Receiver enabled

  - Modem interrupts and USART interrupts disabled

  - INT 09 (vector at 024H) selected for Receive Interrupt Handler.

  - INT 10 (vector at 028H) selected for Transmit Interrupt Handler.

- Program the second USART with similar parameters except that use INT 14 (038H) and INT 15 (03CH) for receive and transmit interrupt handlers respectively.

- Program the programmable interval timer 82C54 with following parameters.

  - Configure Counter 0 as binary counter in mode 2 to generate clock ticks every 10 milliseconds. Note that PCLK jumper on board must be connected to CLK0 input of 82C54.

  - INT 8 (020 Hex) used for timer interrupt handler which passes the clock ticks to VRTX. The OUT0 pin must be wired to IR0 pin of 82C59A.

  - Counter 1 and Counter 2 are cascaded and both are programmed as binary divide by N counters in mode 3. The actual division factor is loaded by the application program.

  - The OUT2 pin is connected to CLK1 pin of PIT and OUT1 pin is wired to IR4 of PIC.

  - INT 12 (030H) is used for odometer interrupt handler. The odometer pulses are fed to CLK2 and are then divided by cascaded counters.

- Program NS 58167 Real Time Clock and Calendar device.

  - Clear all RAM and Registers.

  - Program the device with the current values of hours, minutes, seconds, day, date, month and year as specified by user supplied table.

  - Issue GO command to start the command.

- Create the First Task with following parameters.

  - Use the SC_CREATE system call with id=0 and priority=0 to make it highest priority task.

  - The code of the task must be able to create other task after multitasking starts.
  - Normally it will delete itself after creating requisite tasks.
- Issue VRTX_GO call to start multitasking.
- Interrupt Service Routines
  - USART Character Receive Handler uses UI_RXCHR call to get a character from the I/O buffer and returns in CH.
  - USART Character Transmit Handler uses UI_TXRDY call to put a character in I/O buffer and uses TXRDY driver to send it out.
  - Timer Interrupt handler uses UI_TIMER call to pass on the clock tick to VRTX.
  - Real Time Clock interrupt handler reads the RTC device and returns the requested information about time, date etc.
  - All interrupt service routines are flanked by UI_ENTER and UI_EXIT system calls. All of them have to issue an explicit End-of-Interrupt command to 82C59A PIC to reinitialize it.

5.0 Design of a Multitasking System

The design of a real time system can be described by three distinct steps namely system specifications, hardware development and software development. Each of these categories are very wide in scope and there is a lot of material available on each of them. In this section a brief overview of these activities have been presented with the end application in mind.

5.1 Analysis of System Requirements

It is very important that the system specifications are analyzed in great detail and understood properly before proceeding with any kind of design activity. The system specifications are basically a description of various activities that the system needs to perform under a given situation. At the specification stage there is virtually no concern for the method or means of achieving those requirements.

In the case of a real time system these specifications would typically include the following.

- Functions to be performed by the system.
- Techniques or algorithms to perform special functions.
- The interface between the control computer and outside world.
- The response time or accuracy requirements.
- The operator or human interface considerations.

5.2 Preliminary System Design

The goal of the preliminary system design is to begin to specify the nature of the control flow, data elements and functions in the real time systems [21]. The very first step at this level is to draw a detailed block diagram showing the function level representation of the system along with the interface details. Hence each block in the block diagram should represent a function and each connection should represent flow of control or data between them as well between blocks and outside world.

In order to carry out the preliminary design we need to represent the control and data flow information for the entire system in a concise and comprehensible manner. This has been a favorite topic for debate among real time system designers and there are no firm guidelines available which can demonstrate the superiority of one approach over another. Various tools employed at this level are flowcharts, data flow graphs, algorithmic state machines, modified data flow graphs etc. A good discussion of these topics can be found in [5] and [9].

However the basic idea is to be able to analyze and understand the way various blocks in the system are going to interact with each other and under what circumstances. The utility of these tools is largely dependent on the complexity of the system being implemented. At this stage of design the designer must have control and data flow information explicitly represented along with the interfaces.

5.3 Functional Decomposition

During preliminary design stage a detailed description of the application is prepared using composite data flow diagrams and a block diagram. A list of functions required to realize the transformations appearing on the data flow diagrams is to be made next. To begin with this could be a macro level list of functions. Subsequently each function is to be analyzed in detail to determine the smaller functions that make up this larger function. This process is commonly referred to as functional decomposition. The reader is referred to [7] and [41] for a comprehensive treatment of this and other system design topics.

The objective of this exercise is to identify a set of functions that will be required to achieve the desired performance. This will avoid unnecessary duplication of functions doing more or less same type of job. It also encourages modular top down implementation. A principal goal of functional decomposition is to divide the system up so that it consists of as much possible of functions which are used in several different parts of the system [21].

In order to illustrate the relationship between different functions they need to be represented in some kind of hierarchical diagram. The tree diagrams are a popular and useful choice to document system organization and interface between different components. The high level functions, closer to the root, are basically sequencers which invoke lower level functions and pass information from one function to another [21] representing the control structure of the system.

5.4 Module Identification

The idea behind the concept of functional decomposition is to arrive at some number of independent function blocks. These function blocks are called modules and are largely independent of other modules and interact with them through a well defined interface. The modules can be designed separately and or by separate people but they will fit neatly into the overall system due to these characteristics. Moreover these are independent of data being passed to them and it is recommended that no control information is passed to them.

Mauch [21] identifies following characteristics of a module.

1. The module "hides" the information required to implement the module function from outside. The use of module is simplified since no knowledge of internal operation of module is required. Also the module can be modified without affecting the rest of the system as long as the interface specifications remain unchanged.

2. The links between the module and the rest of the system can be minimized and rigorously defined. These connections and the specifications of data passed constitute the interface to the module as seen by other modules. Ideally only data is passed not the control information.

3. The module is designed to perform a well defined function.

The modular design approach is highly productive and provides for reliable and maintainable system design. It becomes critically important in case of software intensive applications. It also permits simultaneous development of different sections of the software and testing. The ease of design and testing stems from the small size and well defined interface of the modules.

5.5 Division Between Hardware and Software

At this point in the design cycle all the information about the system has been documented and the system requirements have been broken down into modules. The next step is to decide as to which of these modules will be implemented in hardware and which will be realized in software. The decision made at this point will have very significant impact on the overall system development time and cost.

Some functions are more amenable to hardware implementation than software implementation. These could be transducers, analog to digital converters, tachometers etc. Hence all the functions which are inherently suitable for hardware implementation and those that are grossly inadequate for software implementation are grouped together. Similar exercise is to be repeated for modules permitting efficient software implementation.

This process will categorize majority of modules into one of the two possible groups. The remaining functions are to be analyzed further to examine their suitability for one category or another. Various considerations like future upgradability or design and maintenance cost etc. may have to be considered before a final choice is made.

5.6 Hardware Software Tradeoffs

Various kinds of trade-offs, mostly performance versus cost, are an integral part of system design. However the hardware and software trade-offs assume very important proposition in case of a real time systems because of stringent performance requirements. Typically a software implementation of a module is slower than the hardware implementation but the software implementation is more flexible [21]. The cost of implementation will obviously depend on such diverse factors as quantities required, technology or the target system.

5.7 Development of a Multitasking Vehicular Data Acquisition System

The remainder of this section describes the development of a multitasking vehicular data acquisition system using the techniques described above. The objective is to realize the functions of such a system by using a real time executive VRTX and demonstrate the versatility of this approach. The material presented here concerns itself with a well defined application in order to retain the objectivity of the discussion and to illustrate the concepts and techniques used in development of a real time multitasking systems.

5.7.1 System Specifications

The system to be designed has to provide following information.

1. Number of passengers entering the vehicle at each stop.

2. Number of passengers leaving the vehicle at each stop.

3. The time and mileage when the vehicle begins an idle period of one minute.

4. The time and mileage when the vehicle ends an idle period of two minutes or more. The duration for which it was stationary is also required.

5. Number of hours since the system has been powered on or since last dump of logs.

6. The distance since power on or last dump of logs.

7. The vehicle identification number.

8. Hardware fault reporting if detected.

9. System diagnostics invoked by a PC connected to its serial port.

10. Dumping of logs from the system to a PC.

11. Record the change of signboard signalling end of route.

5.7.2 Single Processor versus Multiple Processor Approach

In this application there are eight pairs of sensors which will generate sixteen different interrupts which are to be attended to immediately. In addition to this other inputs to be acquired and processed are signpost input, odometer inputs and door status signals. It is possible to use two single board computers for signals pertaining to each doors and one for the rest of the signals. Alternatively only one single board computer may be used for all the signals if it is powerful enough.

5.7.2.1 Multiple Processor Approach

Both positive and negative aspects of this approach have been enumerated below.

- At least three single board computers, three interface boards and three RS-232C links would be required.
- Some kind of communication protocol and contention resolution scheme may be required for error checking on data being transmitted or received on serial links.
- Relatively low reliability and maintainability.
- Easily expandable to other more complex applications.
- Relatively simpler sensor configuration.
- Can perform in degraded mode.

5.7.2.2 *Single Processor Approach*

The salient features of this approach are,

- The lack of inter-module communication means that entire cpu attention can be devoted to multitasking.
- Higher reliability and maintainability due to fewer components and connections.
- Low power consumption and low cost due to fewer connectors, cables and driver circuits.
- Can be easily customized but poor expandability.
- No degraded performance possibility.
- A high speed processor and support circuitry required.
- The use of backplane bus in case of more than one module increases space and noise overheads.

The application under consideration here is small from the point of view of number of input signals. But the number of input signals do not give a realistic estimate of hardware requirements since the rate at which these input might change is of greater importance.

The existing systems with similar performance requirements use multiple processor approach to address the frequent change in input signals. That is reasonable when there is no multitasking because when cpu is tied up with some activity it is likely to miss some transitions which may be detrimental. It was decided to abandon multiple processor approach in favor of single processor approach for the reasons of reliability, cost and above all multitasking. A preliminary block diagram of this system is given in Figure 22 on page 78.

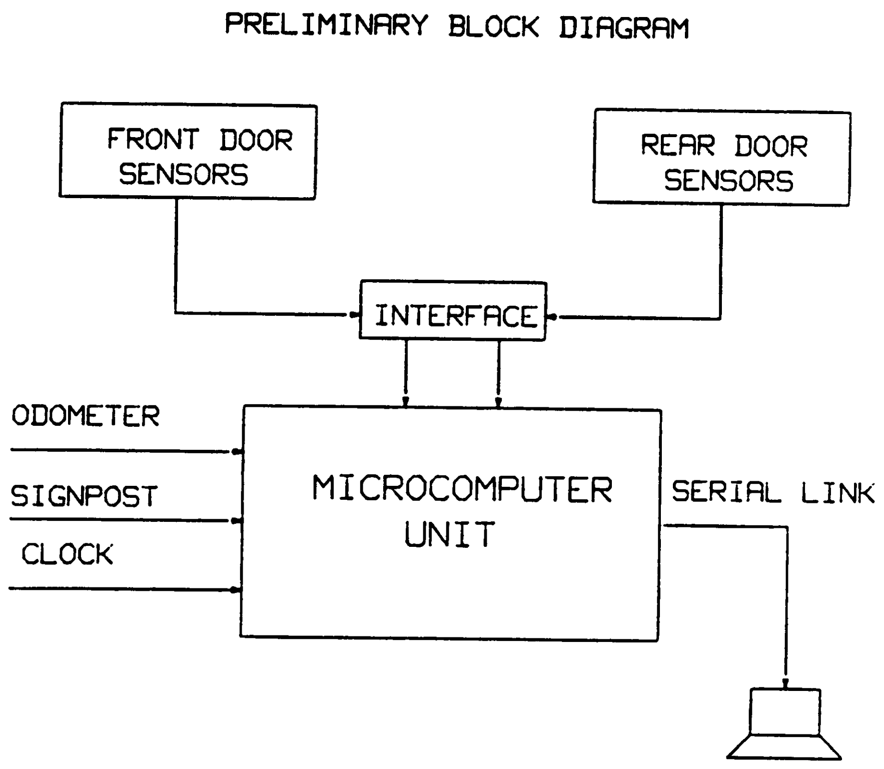


Figure 22. Preliminary Block Diagram

5.7.3 Hardware Selection

The proposed system will consist of a single board computer preferably with CMOS components for low power. The board will contain Hunter and Ready's real time operating system VRTX to manage the resources in the system. The VRTX will be ported on this board to provide a shell or a higher level environment for development of application program.

The microcomputer board chosen for this application is MSI-C988 single board computer from Microcomputer System International. It is a 80C88 STD BUS based board operating at 5 MHz and consists of a programmable interrupt controller, two USARTs with two RS-232C ports, a programmable counter timer and parallel input and output lines. Please see Appendix-B for further hardware details about this. The choice of STD bus compatibility was made with future expansion in mind since a large number of memory and input output cards are available for this bus. A 64K memory board MSI-C764 was also chosen to ensure sufficient memory for VRTX workspace, monitor workspace and application program.

It was decided that a general purpose board will be used to build any interface circuits required to connect field signals to the processor for a given application. A suitable I/O interface card can be bought off the shelf or built if the application is well known before hand. But in this case the main objective was to provide a multitasking vehicle which can be used to implement desired application.

5.8 *Software Development for Multitasking System*

Having decided on the type of hardware as well as the input output interfacing of sensors and the microcomputer the next step would be to concentrate on the software design of the system. The software design techniques and methods are largely the same as in any software system except that now certain activities are to identified which can execute independently. After hardware is completely tested and installed the board support package must be developed for this configuration. The development of a BSP has been discussed in detail in previous chapter.

Dividing the application into tasks is one of the most important aspects of multitasking system designs. Unfortunately there is no one best way to do this and there are no formal methods to achieve it. Consequently this is to be approached on a case by case basis. However there are some general rules and guidelines given in [32] and [9] that can be used as a basis.

Reddy[32] suggests that each functionally different activity be assigned to a different task which can be further subdivided into tasks if it consist of concurrent activities. It is also suggested that a function module may be divided into subfunctions with different inputs and the subfunctions with different priorities are assigned to different tasks. However in DARTS method proposed in [9] the decomposition into tasks is realized by identifying concurrency of various activities on the data flow diagram. The criteria for deciding whether a transform should be a separate task or grouped with other transforms into one task are,

- Dependency on I/O.
- Time critical functions.
- Computational requirements.

- Function cohesion.
- Temporal cohesion.
- Periodic execution.

5.8.1 Analysis

The first phase of analysis deals with the basic mechanisms required for the system irrespective of their availability. The simplest way to analyze an application is to draw a system state transition diagram as suggested in [9] and [39]. A system state diagram is a very generalized representation of various states the system might possibly get into and their interrelationship. Each state is represented by a circle identifying it and each arrow represents a condition causing the transition from one state to another. It gives a high level of abstraction of system behavior and helps in its appreciation.
The state transition diagram for this application is presented in Figure 23 on page 82. Note the level of abstraction, each state is represented in its broadest form and no details are provided. However the transition diagram clearly documents various states of the system and how one can be reached from another. It is also the right place to identify each system activity with either a state or a transition condition. The diagram needs to be modified till it incorporates all required system functions. The preliminary block diagram of Figure 22 on page 78 can be used to cross check this.

This is the starting point for preparing a data flow graph for the entire system. The techniques to draw meaningful data flow graphs have been discussed in great detail in [7], [9] and [39]. No attempt will be made here to explain the procedures to prepare data flow graphs. Since drawing the data flow graph in its entirety may be too cumbersome I suggest that a macro level data flow graph of the system be drawn. This macro DFG will represent the system ac-

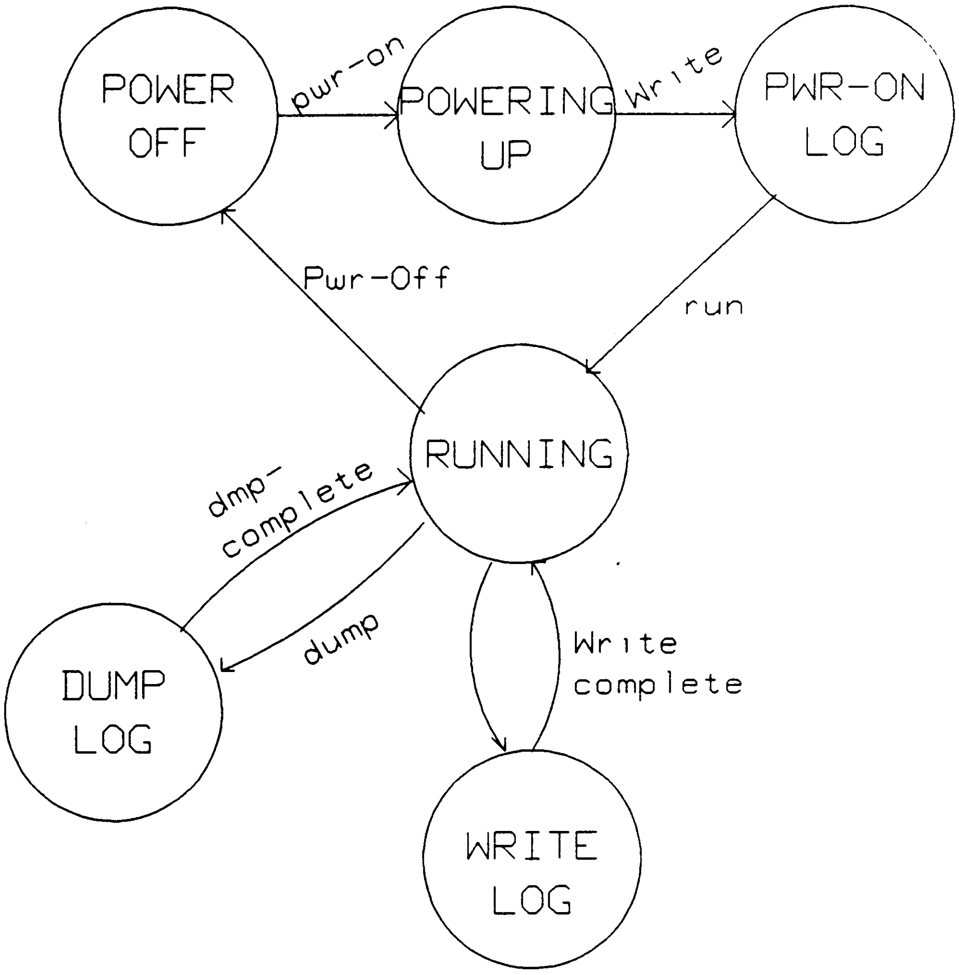


Figure 23. State Transition Diagram

tivity as some definable functions which may or may not be realizable. The idea here is to somehow group similar activities into definable function blocks. Each of these blocks can then be separately expanded till they are broken down into realizable function modules with well defined interfaces.

The Figure 24 on page 84 shows a macro level data flow graph for this application. Note that the DFG for an application is not unique. This macro DFG incorporates almost all of the system functions without getting into details of any one. For example function "Process I/O" is a general purpose function which accepts sensor inputs and transforms them into some form expected by another transform. No details regarding type or quantity of information or the nature of transformation are specified here. The macro level data flow graph provides a structured outline of the system with various functions identified along with their input and output interfaces.

5.8.2 Identification of Functions

At this point an attempt will be made to list those functions according to activities they perform. Each function may perform more than one activity but some activities may require more than one function. From the specifications and the preliminary design steps following conclusions about the functional requirements can be drawn.

1. Detect passenger activity in both front and rear doorwells by monitoring the all eight pairs of sensors continuously.

2. Continuously monitor the open/close status of both front and rear doors whenever vehicle is stationary.

3. Keep track of odometer transitions.

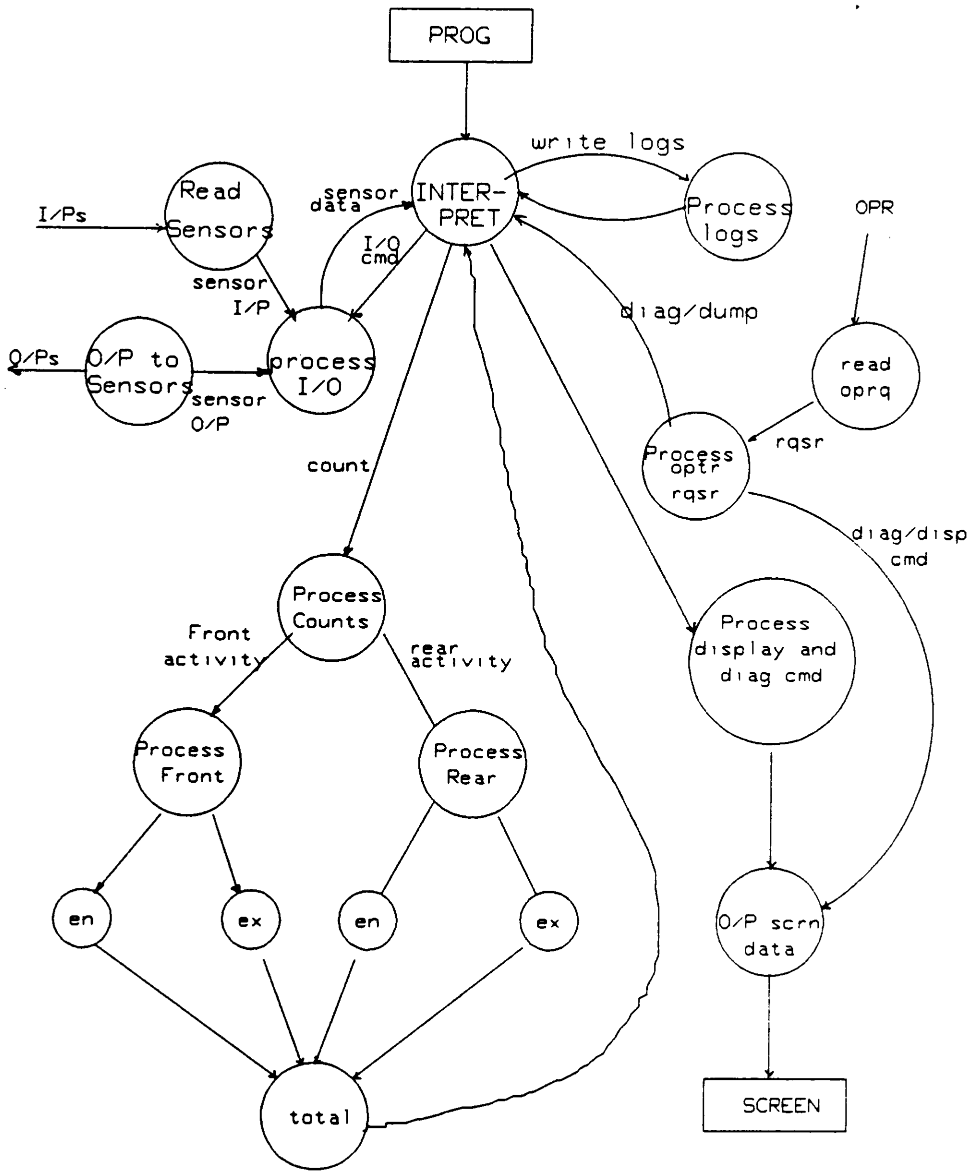


Figure 24. Macro Data Flow Graph

4. Maintain a timer which starts whenever vehicle stops and times out after one minute if not reset by odometer pulse.

5. Maintain a timer which starts two minutes after vehicle has been idle and stops when it moves.

6. Generate and store the data about passenger activities at different stops in form of defined data structures.

7. Also record any abnormal condition.

8. Allow down loading of contents of specified memory area to a PC or other serial device.

9. Permit invocation of diagnostics from PC and provide the requested information.

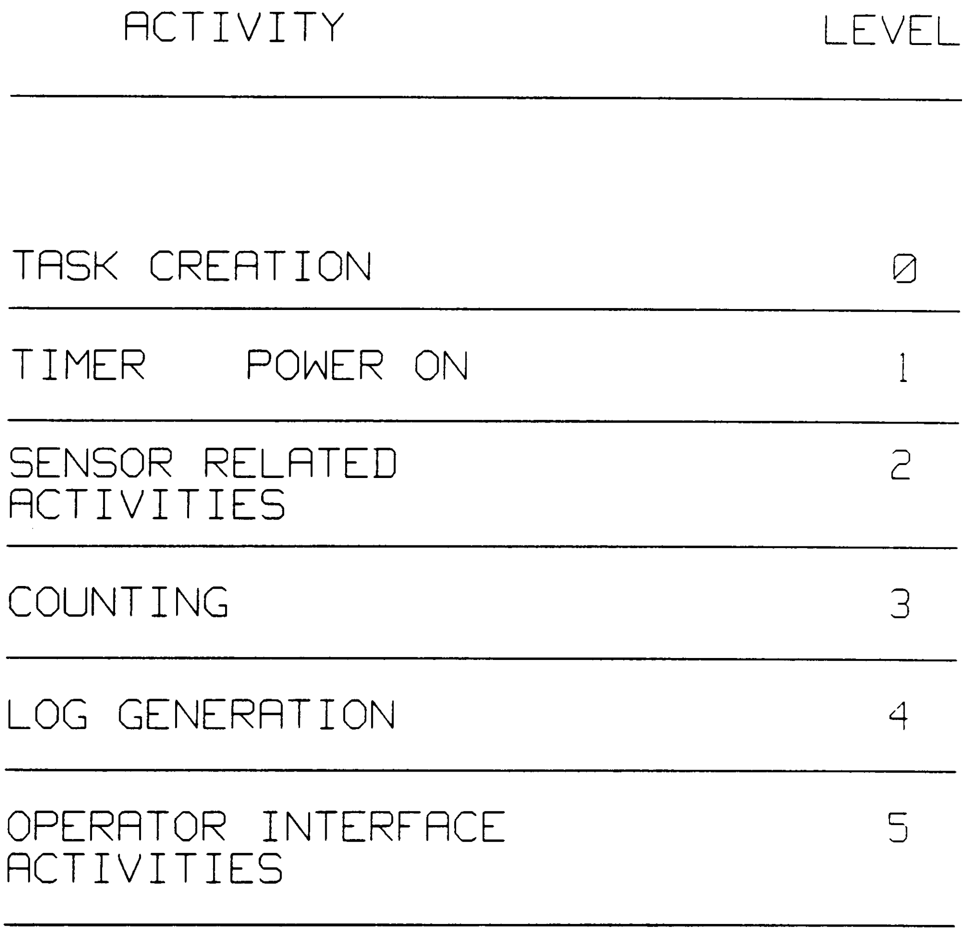
10. Reset the contents and start afresh.

5.8.3 Classification of Activities

Once various activities of a given system have been identified it is time for looking at them in greater detail. In the process described above no mention was made of ordering or importance of these activities. Another important aspect to be considered here is the data dependency of these activities. In any given system various activities are to be performed in certain sequence to achieve the desired results. The ordering of events or activities does not mean sequential processing.

In order to classify these activities each of them is to be considered in detail and in context of the overall application. The idea here is to prioritize these activities in some fashion. This is done by first identifying the most important and critical activities and classifying them as highest level of activities. In terms of the implementation it means that these are to be attended to as and when they demand service. The criteria for identifying highest priority activities could be many but the most important criterion according to Reddy [32] is lack of user control over the activity. This implies that all the activities not under user control must be assigned highest priority level.

The identification of most critical activities provides a good basis to evaluate rest of the functions with respect to this level and then assign them to one of the lower levels. Normally the functions that depend on some other functions to proceed are assigned to a lower level than those they depend on. Generally speaking the functions that handle external activities as well as initialization functions are assigned the highest priority. The macro level data flow graph can be used to examine as to which functions are dependent on other functions. It provides a solid criteria for prioritizing various functions in the system. Another factor considered in classification of these function is the similarity of operation in terms of inputs and outputs. The classification of various functions identified by macro DFG for the given application is shown in Figure 25 on page 87. Note that power on initialization and system timer processes are accorded the highest priority after the task creation module since both are imperative for any subsequent function. The next level has been allocated to sensor handlers since the application has no control over these and if they are not attended to immediately important information may be lost. The diagram groups the activities by the class of services they provide and no specifics of these have been mentioned yet. This only provides us with the information about their relative importance and dependence.



ACTIVITY	LEVEL
TASK CREATION	Ø
TIMER POWER ON	1
SENSOR RELATED ACTIVITIES	2
COUNTING	3
LOG GENERATION	4
OPERATOR INTERFACE ACTIVITIES	5

Figure 25. Classification of System Activities: According to their relative importance and dependence.

5.8.4 Functional Decomposition of Design into Tasks

The decomposition of a design into tasks is an iterative process in the sense that it may be advantageous to restructure the task layout or further subdivide a function. The most important tools required at this stage are the task priority level layout, the macro level DFG, list of functions and activities and detailed data flow graphs of macro functions specified in macro DFG.

The micro data flow graphs of a macro function would provide the details of the internal structuring of that macro function. Note that input output requirements are already known from macro DFG. The activities contained in macro functions can be identified with one or more activities in task priority level layout. If not then these are to be further divided or task layout is modified. The criteria suggested by Gomaa[9] could be used to identify one or a group of activities as potential tasks.

In this case the process was started from the priority level layout. For each function listed at a level a detailed data flow graph was drawn identifying many transforms which could be used directly as tasks. However each such determination has to be verified with macro DFG to ensure proper input output interface. Another important factor taken into account in the process of task definition was the potential for concurrency. It was determined by the relationship of this task with other tasks or functions.
The functional decomposition of the given application appears in Figure 26 on page 89. The size of a particular task or number of tasks for a given activity or subfunction are largely a matter of user preference [5]. However it is recommended that the tasks adhere to specification of "module" as presented earlier in this section.

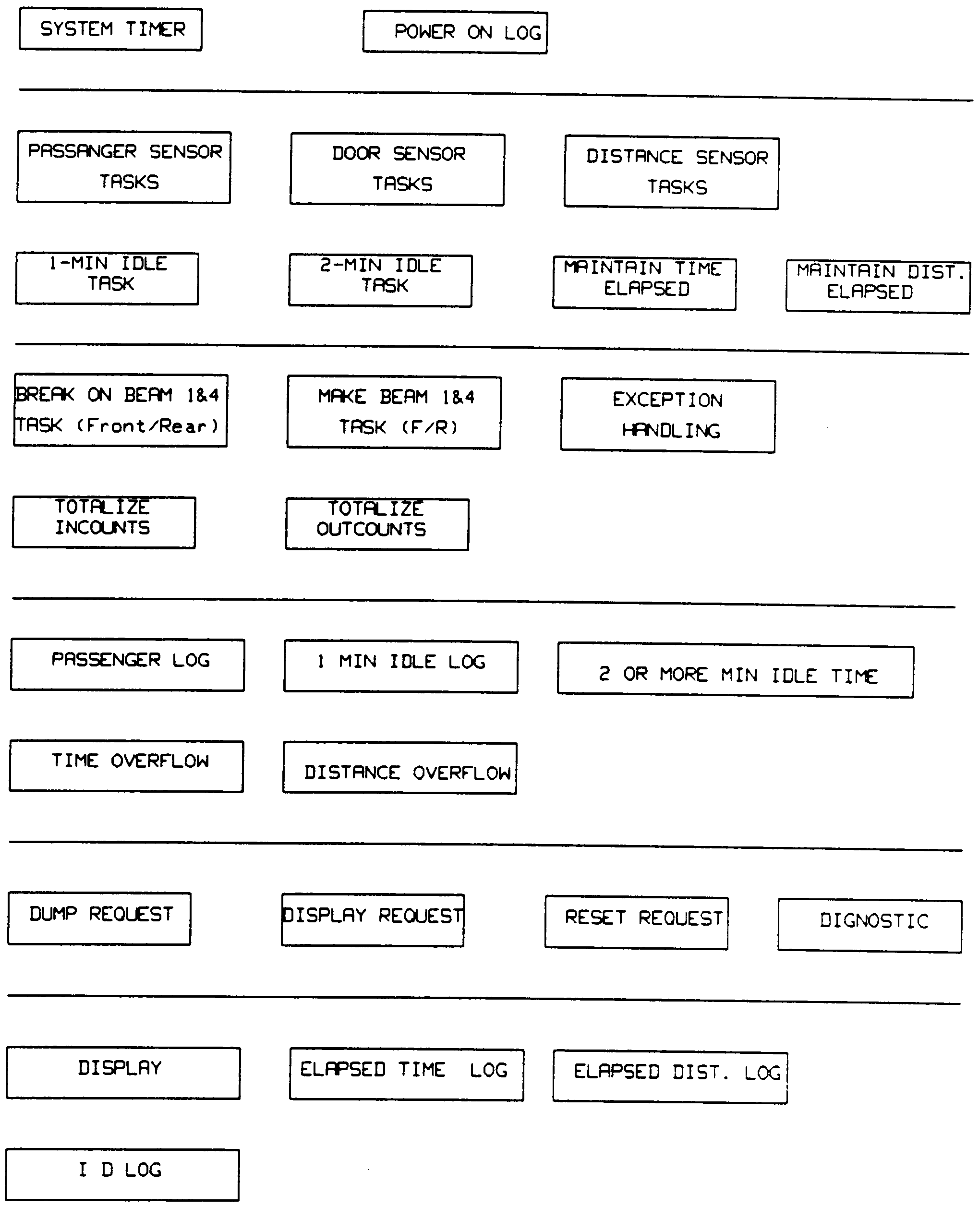


Figure 26. Functional Decomposition of Design into Tasks

5.8.5 Task Identification and Definition

The individual tasks and interface they present to other tasks in the system can easily be identified after the design has been decomposed into lower level functions which accomplish few things. The tasks on the other hand may be viewed as lowest level functions doing just one job. The previous step was necessary to determine the priority levels to be assigned to the tasks in a meaningful way. The task level layout given in Figure 26 on page 89 is used in this case to identify tasks at each priority level and their data flow graphs are then referred to establish their input output interfaces.

All of the functions must be accounted by the tasks being identified. Some duplication of activities is almost unavoidable but structured techniques minimize the risk. The modules which handle external events are normally not defined as tasks to make sure that these events do not go unrecognized. These are collectively called interrupt handlers and operate asynchronously and independently of VRTX scheduled tasks.

Making use of this information the set of activities pertaining to information acquisition and transfer to and from the external world can be grouped as interrupt service routines. They may form a bulk of processing requirements in systems with large number of inputs and outputs. In our case all the functions identified to deal with passenger counting sensors, door control, distance transducer or odometer and signpost input have to be designed as interrupt service routines so that they can respond to external activities in timely manner. A timer interrupt handler is required by VRTX for timing functions.

These interrupt handlers take care of external signals and many activities associated with them leaving fewer number of functions to be dealt with.

5.8.6 Tasks at Priority Level 0

The only task at this highest level is the original task which was explicitly created prior to beginning of multitasking via VRTX_GO call in post initialization phase. This task is designed to create rest of the tasks that may be required to achieve the desired results. This task deletes itself after creating remaining tasks and then these tasks start competing for resources.

5.8.7 Tasks at Priority Level 1

The examination of decomposed function chart of Figure 26 on page 89 reveals that one of the top priority tasks is the power on log generation and another is system timer. The system timer task is scheduled for execution in response to a request from timer interrupt service routine (ISR_TIMER) which handles interrupts from a programmable timer device like 82C54. This task keeps VRTX informed about each timer tick that expires. The power on log is created only once that is at the time of power on and hence is a good candidate for top most priority.

5.8.8 Tasks at Priority Level 2

At the second level the major activities relate to passenger counting, time and distance maintenance and idling tasks. This is also the level at which interrupt handlers can pass information to waiting tasks. However this level has been assigned to distance and time maintenance tasks because all the activity has to be time stamped and most of the logs require distance to be specified. The important tasks at this level are one and two or more minutes idling tasks which monitor the period for which vehicle is stationary. Other task at this level monitor the distance and time overflow.

5.8.9 Tasks at Priority Level 3

The tasks at this level are almost exclusively devoted to counting activities. The tasks at this level are designed to process the information made available by sensor interrupt handler (ISR_SENSOR). Each entry and exit from any of the doors is considered as one complete and independent activity since it results in an on count or an off count. The tasks at this level deal with making and breaking of beams and communicate with each other via mailboxes. All count generating tasks for front and rear door are grouped here.

5.8.10 Tasks at Priority Level 4

The tasks at this level are of relatively low importance in terms of real time response requirements. Most of these are concerned with some kind of data manipulation and housekeeping chores. The counts generated by tasks at previous level are collected here from and totalized to produce total number of on and off counts recorded at previous stop. This information is passed to another set of tasks at further lower level.

5.8.11 Tasks at Priority Level 5

All tasks that deal with generating logs for recording the events are grouped here. The reason for low priority is that they depend on many events to take place before their input requirements are satisfied. They need inputs from totalizing tasks, distance and time maintenance tasks etc.

5.8.12 Tasks at Priority Level 6

The tasks that service the requests from the operator are placed here because the operator commands are too slow and are not likely to disappear like sensor related events. The main task here traps the operator request, decodes it and activates the appropriate task to service that request. These task deal with providing some information about the data being collected to the operator.

5.8.13 Tasks at Priority Level 7

These are the lowest level of tasks and are relatively unimportant because they deal with providing displays and other non time critical information. These are designed to provide services to operator interface tasks at a level above it.

5.9 *Interrupt Service Routines*

The interrupt service routines are an integral part of the system. They are designed to handle external events as and when they occur and hence are not controlled by scheduling mechanism of VRTX. They function asynchronously to the various tasks in the system. In fact these provide most vital inputs to the tasks before they can proceed.

In our system there are eight possible interrupts and corresponding handlers were developed as part of the board support package in some form. These were however modified to suit the specific requirements of the application. These service routines along with their function are listed below. Note that priority of these interrupts depends on the way they are connected to the interrupt controller device.

5.9.1 Timer Interrupt Handler

This is the most important interrupt from the system point of view as many tasks are scheduled on a time basis or use timing parameters. The purpose of this interrupt service routine (ISR) is to keep VRTX informed about expiry of timer ticks. It is done by issuing a system call SC_TIMER from within the ISR.

5.9.2 Odometer Interrupt Handler

The odometer generates certain number of pulses for every unit of distance travelled. These pulses are fed to a divide by N counter and its output is used to interrupt the processor. This ISR posts the arrival of a divided odometer pulse to various mailboxes on which tasks like one_minute_idle or distance_overflow pend. It also maintains its own distance counter which is used by elapsed distance tasks. It is a vital interrupt and hence deserves a higher priority than what is assigned to it in present system (3). It was done because it was known before hand that these will not occur together with those that have been given higher priority.

5.9.3 Sensor Interrupt Handler

This interrupt handler is the most important ISR in the system since it is the only link of the system with the sensors used for counting. The way sensors are setup in this case is that a change in the status of any of the eight beams, four each for front and rear doors, will result in a pulse that will be used for generating an interrupt to the processor. It is the responsibility of the ISR to determine which beam(s) were made or broken and pass on that information to appropriate tasks.

This is shown in Figure 27 on page 96.

The occurrence of this interrupt is an indication that at least one of the beams have changed status. The ISR reads the status of each beam immediately and compares it with the status acquired during last interrupt. An exclusive-oring of these two data items will determine which beam or beams were broken or made. Depending on this determination the ISR has to invoke a subroutine to handle that part of the situation. For example if ISR determines that beam 4

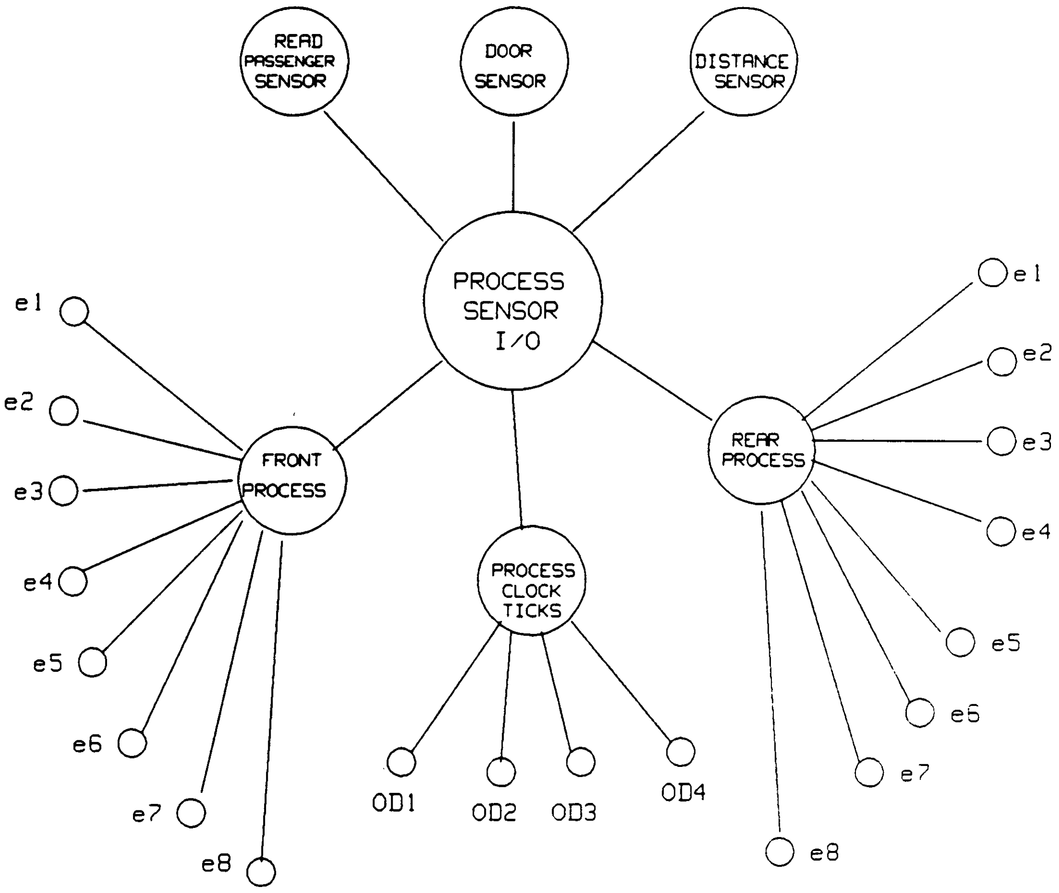


Figure 27. Sensor Interrupt Handling

on rear door was broken and exit flag is set then it is an error condition since beam 4 has already been broken before and has not been made yet. But if exit flag was reset then ISR will set that flag and post appropriate messages to the mailboxes.

The beam status is passed to each pending task as a message and the corresponding sub-routine picks the status of the individual beam and posts to that mailbox. The tasks like break_on_beam1 pend on mailboxes labeled e1, e2 etc. indicating an event. The ISR posts to them when the condition (make or break) is detected.

5.9.4 USART Interrupt Handler

The USART interrupts are handled via two ISRs one for character receive and another for character transmit. These have been discussed in great detail in previous chapter.

5.9.5 Signpost Input Interrupt Handler

This interrupt occurs when the signpost is changed by the operator at the end of route. This may be used for generating another log indicating a new start. This information is used by analysis software that analyzes these logs for variety of conclusions.

5.10 Count Generation Algorithm

Before describing the tasks individually it is necessary that the method of count generation is understood properly. The count generation has been identified as one complete function which in turn was divided into various tasks. These tasks monitor the make and break conditions on each beam corresponding to each door. The concept of in (or on) counts and out (or off) counts is crucial here.

There are basically two types of activities that can result in generation of one of these types of counts. All other activities do not produce any count. This gives us a basis to classify certain possible sequence of events as illegal and the system may ignore them or generate some kind of message. The valid scenario is the one in which either an on or an off count is generated. Since both doors are identical for this purpose it is sufficient to consider only one.

It is assumed that all beams are in made condition to start with. If the first beam to be broken is beam 1 (belonging to outer most sensor) then there is a possible chance of in count. At this point we have two options, one which will require monitoring of remaining beams to be broken in that order (1, 2, 3, 4) to be a valid half in count and another half coming when beams are made in that order. Alternatively if beam 1 was the first beam to be broken and then it was determined that all beams did break and beam 1 did make too then it is a sufficient criterion for deducing that remaining beams will also be made.

The second approach was chosen over the first to reduce the overhead and to filter out invalid scenarios which have to be dealt with when handling each beam separately. Another factor in favor of this method the layout of the sensors. They have been kept in pairs where each is kept very close, a couple of inches, and two such pairs are mounted very closely. This ensures that if a break on one is detected there is a high probability that it will occur on next one immediately.

The same method is used to determine the off counts except that this process starts with beam 4 (inner most sensor) and then proceeds exactly as the previous one.

The count generation process has been illustrated in Figure 28 on page 100.

Also see Figure 29 on page 101 for count accumulation and passenger log generation processes.

5.11 System Handling of Events

This section describes various situations that may arise after the system has been powered up and the response of this system to these externally stimulated events using the tasks identified in the previous sections. The detailed description the tasks that are required to implement the functions of a vehicular data acquisition system can be found in Appendix-D. The discussion includes almost all of the critical tasks and traces their design and functioning. The source code for assembly language implementation of some of these tasks is provided in Appendix-G.

Please note that this discussion attempts to describe the functioning of a task not actual coding. Hence the reader is advised against trying to find one to one corresponds between the description here and the actual implementation. For example it may be advantageous to implement a broadcasting mechanism instead of posting the same message to a large number of mailboxes in practice, but in a discussion like this we are concerned with the condition which results in suspension of the task and the condition which reactivates it.

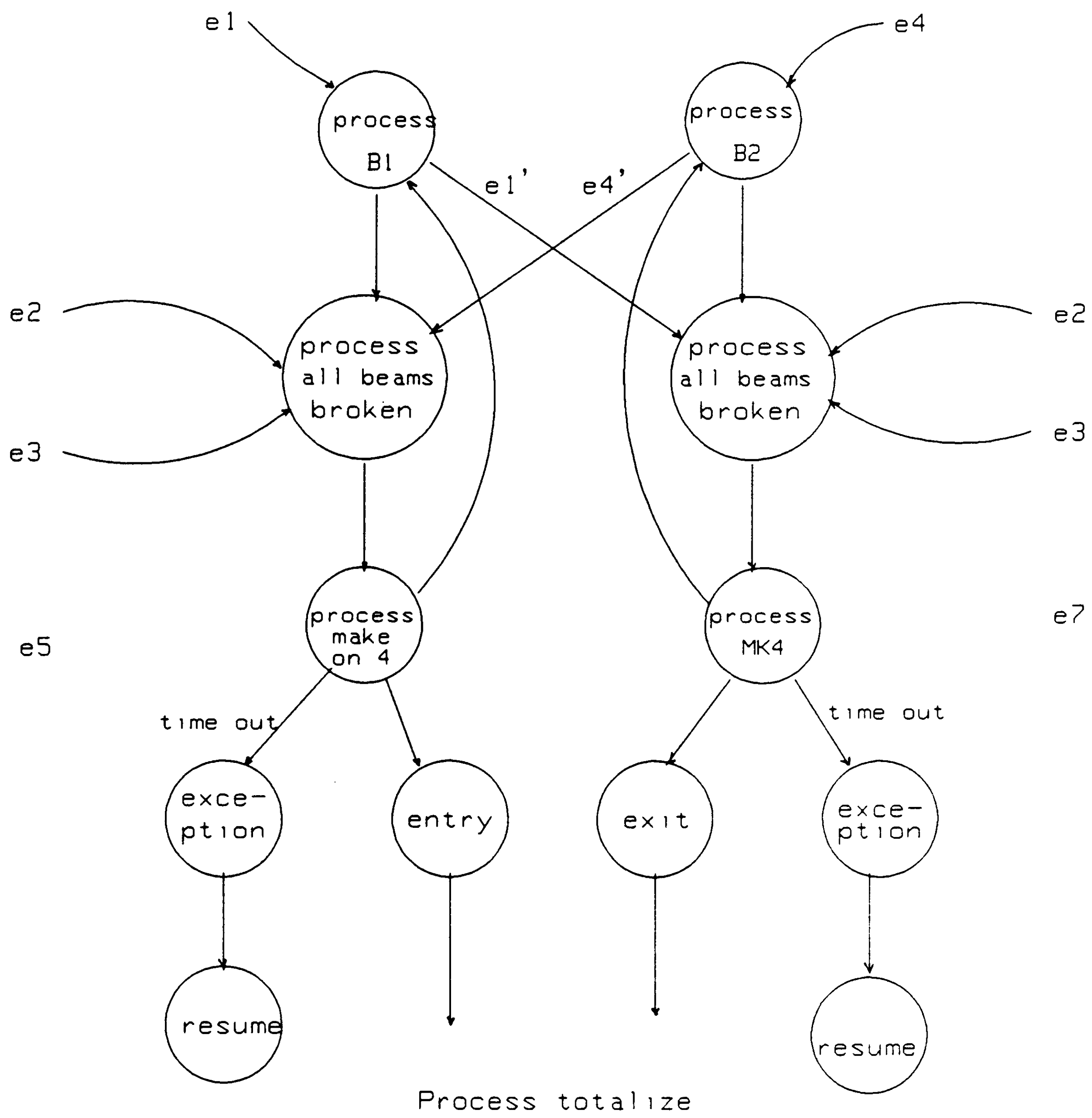


Figure 28. Count Generation Process

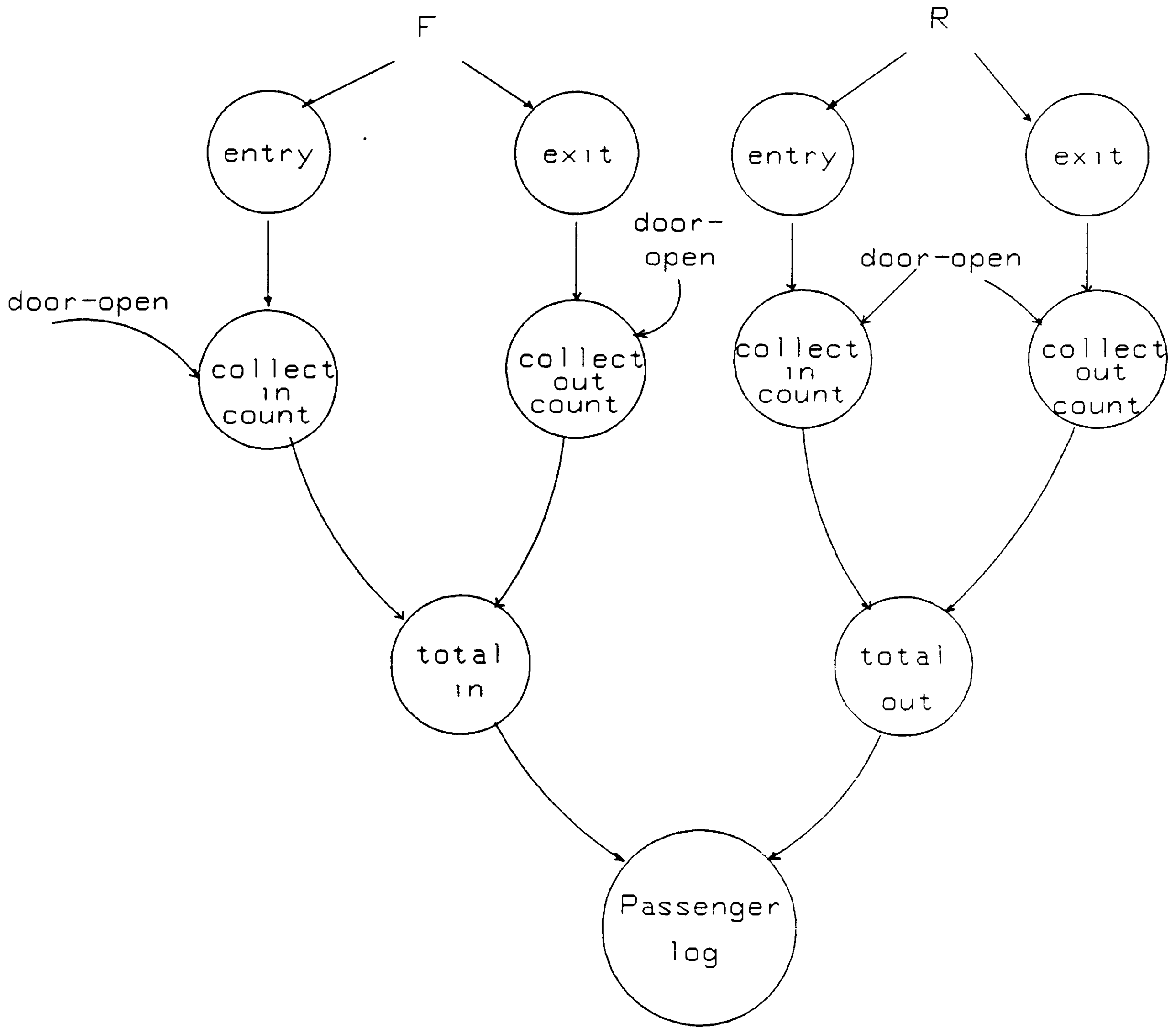


Figure 29. Passenger Log Generation Process

5.11.1 On and Off Count Generation

The count generation is directly related to the sensor activity. As discussed in previous section the sensor interrupt handler posts the status of different beams to corresponding mailboxes on which various make and break monitoring tasks pend. For example Sensor interrupt handler posts to mailbox e1 when beam 1 is broken and it was the first one to be broken. This activates the task break_on_b1 which passes this information to another mailbox for a possible exit scenario. It also commands all_beams_broken task to start looking for break condition on all beams on that door. Note that sensor handler also posts the beam status to this task. Once it is established that all beams were broken after beam 1 is broken the only pending condition is make on beam 1 to generate an in count. The make on beam 1 is indicated by the sensor handler to make_on_b1 task which in turn informs the break_on_b1 as well as count totalizing task.

Consider a possible situation where a person is trying to enter the vehicle from the front door. In order to simplify the situation for a better understanding of the interaction and interplay of different tasks let us assume that no other entry or exit is being attempted. Subsequent discussion will show that the more complicated situation situations can be handled with the same ease. In case of the simplified situation the beam 1 on the front door will be the first to be broken. This will cause an interrupt and the sensor interrupt handler will read the status word giving the present status of all eight beams.

This status word will differ with the previous word in only one place corresponding to beam 1 on front door. Hence ISR_SENSOR will conclude that a break-on-beam1-of-front-door has occurred and it will post a non zero message to the mailbox labeled e1 and no messages are posted to any other mailbox at this time. This ISR will also check the ENTRY flag before posting any message. In this case however the flag was reset so it will be set so that further interrupts can be tracked properly. The ISR exits after that.

The task Break_on_Beam1(F) is responsible for monitoring the breaks on beam 1 of the front door. Note that suffix F is dropped in later references to improve clarity. This task pends on the mailbox e1. After the ISR_SENSOR posted a message to this mailbox the suspending condition of this task is removed and it will be scheduled to run. This task examines whether or not ENTRY flag is set and proceeds only if it was set otherwise it treats it as an invalid sequence. In this case it will go ahead and post a message to All_Beams_Broken (F,EN) task which monitors all four beams and responds with a non zero message when the condition is met. There are four such tasks for entry and exit situations at both doors. Note that ISR_SENSOR posts messages to this task informing it of status of remaining beams.

Once the condition all-beams-broken is detected it will pass on this information to another task Make_on_Beam1(F) which looks for makes on beam 1 at front door. This task at this point knows that all beams have been broken, beam 1 was the first beam to be broken and the entry flag is set. Hence as soon as it receives a make on the beam 1 it signals generation of an on count to a count collecting task. It also posts a message to Break_on_Beam1 task informing it the completion of a sequence which causes the entry flag to be reset and the task is now ready to handle another entry or exit.

This situation has been depicted in Figure 30 on page 104 where each small circle represents a mailbox and the larger circle represents a task. The incoming arrows correspond to pending conditions and outgoing arrows represents the posting conditions. The Figure 31 on page 105 shows an identical sequence of events at the rear door.

A similar analysis can be done for the exit situations which proceeds in exactly the same fashion except that this sequence of event is triggered when the first beam to be broken happens to be beam 4 corresponding to inner most sensor in a doorwell. The front and rear door exit sequences have been shown in Figure 32 on page 106 and Figure 33 on page 107.

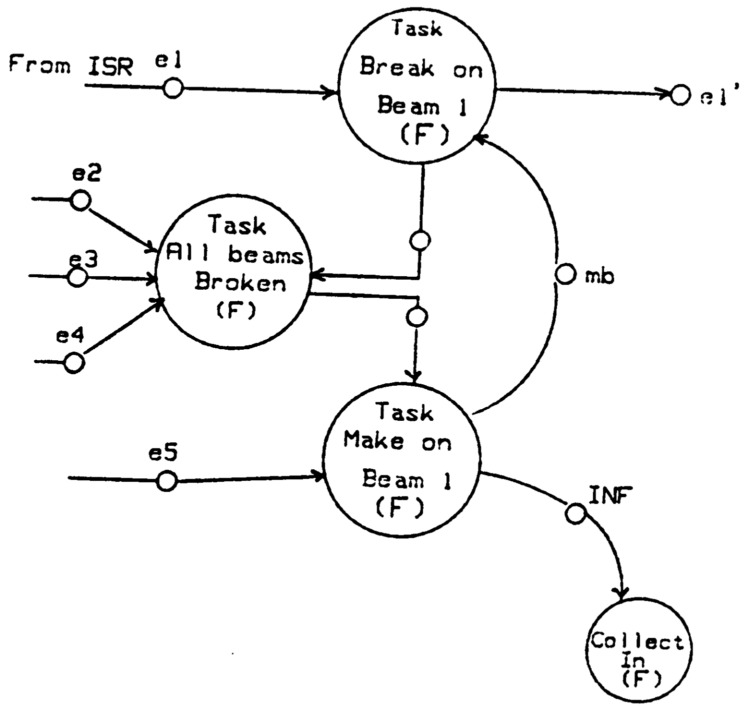


Figure 30. Entry from the Front Door

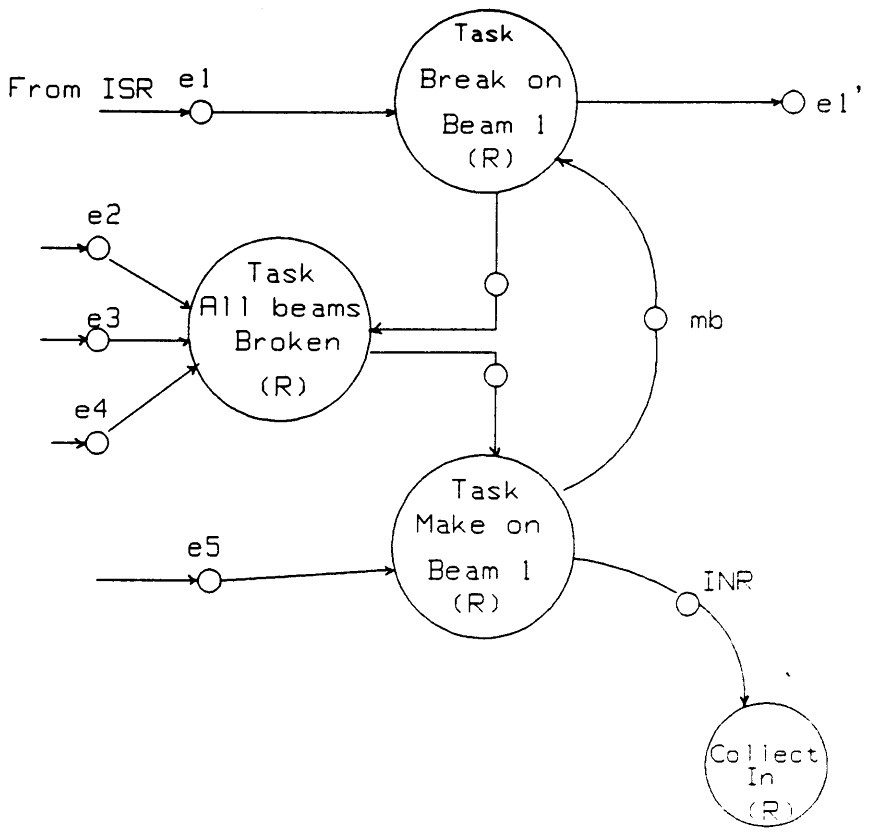


Figure 31. Entry from the Rear Door

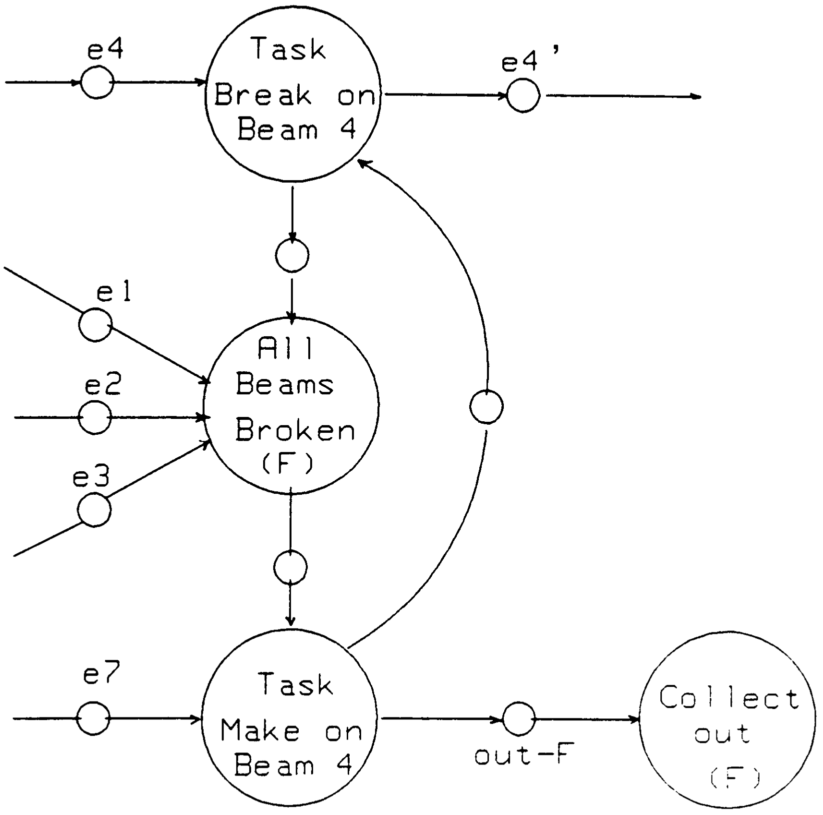


Figure 32. Exit from the Front Door

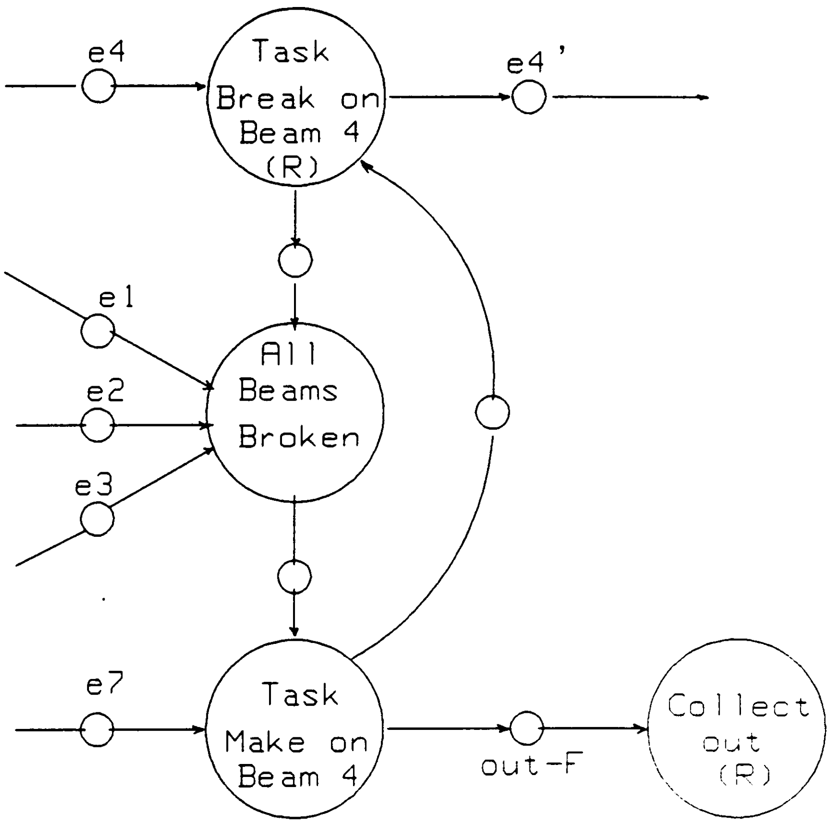


Figure 33. Exit from the Rear Door

This discussion illustrates the procedure adopted to determine if a valid on or a valid off count was generated. It is fairly obvious that all the four tasks participating in this sequence of events were executing concurrently and were being synchronized by message passing between them via mailboxes. Let us consider a more complex situation where a person is entering from the front door and another is exiting from the rear door. In this case ISR_SENSOR will post a break condition to two mailboxes e1(F) and e4(R) activating tasks Break_on_Beam1(F) and Break_on_Beam4(R) which will post messages to their respective All_Beams_Broken tasks handling entry and exit respectively. These in turn will pass on this information to respective Make tasks which will produce the on and off count.

It is evident that these two events were running in parallel and even though both acquire data from same ISR nothing else was shared. These two events need not be synchronized and may or may not get completed in equal time.

5.11.2 Passenger Log Generation

A passenger log is generated if there was some passenger activity at the immediately previous stop. It is created only after the vehicle moves after being stationary at the stop. Hence the first condition for generating this log is that there must have been some in and or out counts produced. The task completing the last activity of the sequence will signal to a collection task availability of a count. The collect_in task accumulates the on counts being generated at one door. A similar task collects off counts. These counts are then passed on to count totalizing tasks which produce the net on and off counts for the vehicle at the previous stop.

The on and off count collection tasks qualify the signalled count for its validity by checking the corresponding door status signals. No counts can be generated while vehicle is in motion. The door signals are monitored by requesting a status read at the time counts become available. Note that while recently generated count is being verified the break and make monitoring tasks are at work looking for new entry and or exit. The collection tasks post the qualified counts in a queue which is read by a totalizing task.

The passenger log is created by a task which pends on vehicle moving condition. As soon as the vehicle moves this task will accept the counts posted to it by the totalizing tasks. In order to complete the log it will also accept the incremental distance and time information from distance and time control tasks.

5.11.3 Idle Time Control

Another important system requirement for this kind of systems is idle time control. This means that the system should be able to monitor the time for which vehicle was stationary as well as the time for which it was moving. In the case of system being considered here the two such situations are one minute idle and two or more minute idle. This mandates that system should be able to differentiate between the two activities.

The present implementation identifies a task just to monitor an idling period less than or equal to a minute. This task continues to look at a particular mailbox which is posted to by odometer ISR. The task is allowed to pend only for a minute on this mailbox and hence when the vehicle is stationary for more than a minute it will return with a time out informing in effect the expiry of one minute idle period. This condition is passed to a one_minute_idle_log task which creates and writes the log.

Similarly a two or more minute idle period is detected by first waiting at a mailbox for two minutes and in case of time out waiting at another mailbox till odometer ISR posts a message to it. Both mailboxes are posted to by odometer ISR. After detecting the movement it reads VRTX maintained time and computes the time for which it pended giving the total idling time. This information is passed to two_minute_idle_log task which appends distance and time information and writes the log.

5.11.4 Overflow Control

There are two tasks in the system which are exclusively dedicated to keeping track of time and incremental distance information respectively. Note that this information is used by each task that writes a log since it is to be included with almost every log. These tasks start counting the time and distance in incremental units by pending at odometer posted mailboxes for distance and by using VRTX sc_tdelay call for time. Each write log operation resets the accumulated value.

The distance overflow log task is activated by distance overflow task whenever the distance accumulated by it exceeds the value that can be held in a byte. It generates the incremental time information and writes the log. The hour overflow log task is activated by hour overflow task whenever the ticks accumulated by it exceeds the value equivalent to one hour. It generates the incremental distance information and writes the log.

5.11.5 Operator Interaction

The tasks organized under this category are designed to address to various requests that can be made by operator. These could pertain to dumping, displaying or resetting the logs. Most of these tasks are low priority tasks and are given control only when sensor related tasks are dormant.

The operator requests and instructions are handled by Get_Command task which depends on USART interrupt handler for its activation. The interrupt handler of USART posts the character received from the operator console to this task. This task pends at a mailbox in which character is passed to it. It examines the character to be one of the valid commands and an error is signalled if not. Otherwise this task will activate a lower level task to handle the request by posting appropriate message to the mailboxes at which they pend.

Appropriate service tasks are activated in response to a valid operator command. For example get_command task will post a non zero message to display_log task which displays the logs on a terminal in response to a request from the operator. It pends on a mailbox which is posted to by get_command task whenever it detects display command character. This task responds by sending out the logs in 25 byte blocks (5 logs) to the I/O buffer and transmit interrupt handler does the rest. Similarly tasks dedicated to resetting logs or dumping logs can be activated by get_command task to handle operator requests.

The operator can also command the system to operate in diagnostic mode to get an inside look at system parameters of interest to the operator. The data provided includes beam status for both doors, door status and current incremental count values. This task will activate tasks at lower level to transmit the information about these parameters. Note that this information is primarily supplied by interrupt handlers or obtained by reading the I/O ports.

5.11.6 Elapsed Time and Distance Control

The system is also required to maintain total time and elapsed and distance covered since the last log dump or the power on. elapsed_time task maintains a count of time ticks since the last time logs were dumped or power on. The purpose of this task is to determine the total amount of time that elapsed since the previous dump. This information is used when the logs are dumped and another log will be created containing this data.

The elapsed_distance task maintains a count of odometer transitions since the last time logs were dumped or power on. The purpose of this task is to determine the total distance that was covered since the previous dump. This information is used when the logs are dumped and another log will be created containing this data.

5.12 Summary

The discussion presented above covers just about every situation for which certain response was part of the specifications. The most important situation from the system point of view is the count generation and recording. However time control functions are equally important as they provide the basis for any meaningful analysis. The system response for situations not discussed above can easily be predicted by drawing a data flow graph leading to that situation. Most of the pending situations have been implemented with time out provision to avoid deadlocks and dead waits. This was done so because in real time each event has an upper bound within which it should be serviced and if that point is passed then the response is useless. In the approach here the requesting task is intimated of failure so that it might take alternative action.

6.0 Conclusions

The objectives of this thesis were manyfold. The most important of them was to gain a first-hand understanding of the complete design process involved in developing a real time multitasking system using off the shelf hardware and silicon software components. In order to pursue these goals a hands on approach was taken and a real time multitasking system was built from scratch. It also provided a vehicle to test various design techniques proposed in earlier thesis works and encountered in literature. The VRTX/88 a silicon software component which is a real time executive kernel was ported to a STD bus based 80C88 microcomputer board and an actual real-time application was implemented.

Probably the most important and definitely the most enlightening activity involved in this thesis work was the porting of VRTX/88 kernel onto an off the shelf board from the scratch. The porting of an operating system refers to development of certain specialized device and system dependent code in order to establish the logical and physical interface between the hardware and the kernel. This turned out to be a difficult phase but that was mainly due to inadequate hardware documentation and a difficult environment for embedded microprocessor software development. An IBM PC was used as the development station for developing the board support package as well as the application software.

Due to memory constraints TRACER, a silicon software component for debugging support and a companion product of VRTX/88, was not installed. It made the software development more difficult and time consuming especially the application software. However the need for a good debugging tool is most vital during porting of the operating system when the designer has no means to look inside the system. TRACER is to be installed along with the VRTX to be able to provide debugging support. The linker available on the IBM PC produces only '.EXE' file. It is necessary to have a software package (locater) which can transform this .EXE file into a relocated file with absolute addresses, ready to be downloaded and executed. A real time emulator would possibly be the best choice at this stage of integration. But most 'real life' real time system designers are more likely to encounter PC based development environments hence a good investment of time in identifying and selecting proper software tools for various activities will pay off handsomely.

The porting of VRTX/88 kernel required a detailed knowledge of the hardware and development of a board support package which must contain initialization routines for all I/O devices in the system including special purpose devices, address information about RAM and VRTX PROM as well as the initialization of interrupt vectors. The interrupt handlers for different input-output devices are also included. A test program was developed to exercise all the devices and their interrupt handlers to ensure that the board support package was working fine. This test program is modeled after a similar program provided by Hunter and Ready in Board Support Package Manual and uses some of their routines. It was at this stage that most difficulties were encountered. The debugging was difficult because it was not certain at any point of time whether the problem is due to hardware and or due to software being tested. Often it turned out to be the hardware related. But once the board support package was debugged and installed it became very convenient to use the VRTX services.

The initial board support package was substantially extended to add a number of tasks to study and to familiarize myself with various system calls and system behavior. It was observed that system became sluggish as the number of tasks increased or when interrupts were allowed to come in at a faster rate. The relationship between the 'performance' of the system and various possible factors contributing to it is not well defined and is a potential research topic. Some of the more important factors could be number of tasks, coding of tasks and interrupt handlers, task design, task identification, interrupting devices and their servicing requirements, and frequency of interrupts. During the development of this application it was observed that an increase in number and complexity of tasks directly degrades the performance.

It was found necessary to build monitor style data structures using system post and pend calls to a mailbox (semaphore) for most of the common resources like CRT, character I/O buffer etc. The VRTX contrary to my initial impressions does not provide an implicit mechanism which allows a given task to reserve the resource till it is either finished or interrupted by a higher priority task. This was noticed during the application software development when a task reading a string from the console lets another task steal a character from this since it was also pending for a character from console.

VRTX/88 however is a powerful and easy to use silicon software component. No knowledge of its source code is required to use the system primitives or to even extend it by adding user supplied extensions. I was able to attach special drivers to augment the VRTX supplied character support during testing. I also implemented my own character I/O buffer using second USART driving another terminal. The single biggest advantage of VRTX/88 in my opinion is that it provides a common and well defined set of mechanisms to support multitasking. These facilities can easily be used by the system designer without concerning himself with the actual implementation of the same. This promotes development of disjoint software modules which connect with each other through well defined links implemented by using VRTX supplied data structures like mailboxes and queues.

Another aspect of this thesis dealt with the implementation of various functions that are provided by automatic passenger counters and vehicular data acquisition systems. In fact this was chosen to be the representative application. This required a detailed understanding of such systems. The necessary information about system activities, priorities and various responses was obtained from previous thesis works and product literature. However the most valuable knowledge was gained by studying and analyzing a real product from Red Pine Associates. This phase of the thesis was hard and slow as too many measurements and observations were required to understand the actual event-response relationships. The information gathered in this phase was used to draw basic specifications for the new system.

The architecture of the new system is entirely different from that of Red Pine system both in hardware and software. The system was not duplicated. The new system took multitasking approach using a single board computer whereas the Red Pine's approach called for use of four microcomputers each dedicated to one particular type of functions. The sensors for the testing of the new system were simulated using switches, pushbuttons, function generators etc. The new system uses a new log structure, the format in which events are recorded, which is more efficient since it provides more information. It became possible by using a real time clock and stamping events with real time. It is also possible to keep track of dates and months if required.

The structured software design techniques were adhered to as strictly as was possible for the application software development. But occasionally I subdivided a given task or combined some of them to generate compact and fast executing code. Moreover it is not always possible to establish one-to-one correspondence between the task identified by decomposition process and its actual implementation. The implementation may sometime redefine the task interfaces.

Any real time system will require some kind of sensors to function and perform in a desired fashion. It was noticed during the development of this application that sensor interface can pose a tough set of problems. The sensor interface is to be designed very carefully since most of these operate on interrupt basis and frequent interrupts will surely degrade the performance of any system, multitasking or not. For example the Red Pine system produces interrupts on every beam disruption. In such a case there will be sixteen interrupts corresponding to make and break conditions on each of eight beams. This will entail quite an overhead. In our case we combined all these interrupts into one single interrupt to save on system entry and system exit overhead incurred by VRTX. It is recommended that more sophisticated sensors be used to reduce the overheads. The fiber optic sensors are a good choice since they can provide processed information to the computers.

I strongly feel that the weakest link in the entire system is sensor related hardware and hence considerable amount of time and research must be devoted in designing or selecting proper kind of sensors. Another important thing to be considered is the software development environment. An additional 64K memory board should be added to this system which can house VRTX and TRACER PROMs as well as the RAM area required for both these components which is substantially large for any useful application. The basic software tools for debugging, locating, loading, disassembly and downloading must be available. It will be nice to have ROM simulators and emulators though they are not absolutely necessary. It is recommended that this system should be developed in full after the initial specifications considered here have been expanded to suit the actual requirements. It is also recommended that new types of counting algorithms are considered for array of sensors which could lead to low level object recognition by profile matching. The study of VRTX performance under varying conditions is a potential topic as it will yield critical clues to the relationship between scheduling algorithms, context switching and the system performance.

BIBLIOGRAPHY

1. Andrews, G.R. and F.B. Schneider, "Concepts and Notations for Concurrent Programming," Computing Surveys, pp. 3-43, v 15, n 1, March 1983.

2. Baker, T.P. and G.M. Scallon, "An Architecture for Real-Time Software Systems," IEEE Software, pp.50-58, May 1986.

3. Brinch-Hansen, P., Operating system Principles, Prentice-Hall,Inc., Englewood Cliffs, New jersey, 1973.

4. Bunce, P., "Silicon Operating System Modules Aid Realtime Control," Computer Design, pp.203,204,207,209, November 1982.

5. DeBrunner Linda S., Real Time Multitasking Systems, Virginia Polytechnic Institute and State University M.S.E.E. Thesis, Blacksburg, Virginia, 1986.

6. "Dijkstra Semaphores Application Note", Hunter and Ready, Inc., August 1983.

7. DeMarco, T.,Structured Analysis and system specification, Yourdon, Inc., New York,1979.

8. Getting started with Silicon Software components version 3, Hunter and Ready, Inc., Palo Alto, California, 1984.

9. Gomaa, H., "A Software Design Method for Realtime Systems," communications of the ACM, v 27, n 9, pp.938-49, September 1984.

10. Harris CMOS Digital Data Book, Harris Corporation.

11. How to Write a Board Support Package for VRTX version 3.0, Hunter and Ready, Inc., Palo Alto, California, 1984.

12. Hwang Kai and Briggs, Computer Architecture and Parallel Processing, McGraw Hill Book Company

13. iAPX 86 Users Manual, Intel Corporation, Santa Clara, California.

14. iAPX 86 Users Manual - Programmers' Reference, Intel Corporation, Santa Clara, California.

15. IOX/86 User's Guide version 3, Hunter and Ready, Inc., Palo Alto, California, 1984.

16. IBM PC DOS 3.1 Users Manual

17. Johnson, C.D., Microprocessor-Based Process Control, Prentice-Hall,Inc., Englewood Cliffs, New Jersey, 1984.

18. Kingsley, Stuart A., Distributed Fiber Optic Sensors - An Overview, Proceedings of SPIE Vol 566, Fiber Optics and Laser Sensors III, 1985.

19. Kuenning, G.H, "Designing Real-Time Software Systems," Sigsmall Newsletter, pp.34-39, v 7, n 2, October 1981.

20. Kuenning, G.H., "Minimal Multitasking Operating Systems for Real-Time Controllers," Sigsmall Newsletter, v 7 n 2, pp.20-27, October 1981.

21. Mauch Konrad and Lawrence P., Real Time Microcomputer System Design - An Introduction, McGraw-Hill Book Company.

22. Microsoft Macro Assembler User's Guide and Reference Manual, Microsoft Corporation, 1984.

23. Milenkovic Milan, Operating System Concepts and Design, McGraw-Hill Book Company.

24. MSI-C988 Microcomputer Card User's Manual, MSI International, Baton Rouge, Louisiana.

25. MSI-C764 Memory Card User's Manual, MSI International, Baton Rouge, Louisiana.

26. MSI-C988 Monitor User's Manual, MSI International, Baton Rouge, Louisiana.

27. "Multi-Processor Applications Using VRTX", Application Note, Hunter and Ready Inc., Palo Alto, California, May 1984.

28. PCLOCATE Users Manual, ALDIA Systems, Arizona.

29. Peterson James L. and Silberschatz Abraham, Operating System Concepts, Addison Wesley Publishing Company.

30. Ready, J. and G. Funk, "Software components create plug-in OS," Electronic Design, pp. 129-138, April 19, 1984.

31. Red Pine Vehicle Data Acquisition System Manual, Red Pine Associates, Ontario, Canada.

32. Reddy, S.T., Multitasking for Sensor Based Systems, Virginia Polytechnic Institute and State University M.S.E.E. Thesis, Blacksburg, Virginia, 1985.

33. Rolander T. and Adams George, Design Motivations for Multiple Processor Microcomputer Systems, Computer Design, March 1978.

34. Savitzky, S.R., Real Time Microprocessor Systems, Van Nostrand Reinhold Company, New York, 1985.

35. Shanker S., Analysis of Microprocessor Based Vehicular Instrumentation, Virginia Polytechnic Institute and State University, M.S.E.E. Thesis, Blacksburg, Virginia, 1985.

36. Takara Ken, Program Design Using Pseudocode, Dr. Dobb's Journal, March 1984.

37. Tseng, V., Microprocessor Development and Development Systems, McGraw-Hill Book Company Limited, New York, 1982.

38. VRTX/86 User's Guide version 3, Hunter and Ready,Inc., Palo Alto, California, 1984.

39. Ward, P.T., "The Transformation Schema: An Extension of the Data Flow Diagram to Represent Control and Timing," IEEE Transactions on Software Engineering, pp.198-210, v SE-12, n 2, February 1986.

40. Yelvington, P. "Embedded uP Applications Require a Real-Time Operating System," Digital Design, pp.50-54,71, January 1983.

41. Yourdon, E. and L. Constantine, Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design, Prentice-Hall,Inc., Englewood Cliffs, New Jersey, 1979.

42. Zarella John, Microprocessor Operating Systems, Microcomputer Applications, Sun City, California.

Appendix A. Analysis of a Vehicular Data Acquisition System

The material in this section deals exclusively with the functionality of the modules supplied by M/S Red Pine Associates for Vehicular Data Acquisition System. The system received from Red Pine was tested and analyzed in detail to gain meaningful insights into its functioning.

A.1 System Description

The vehicular data acquisition system (VDAS) as supplied by Red Pine Associates consists of four modules interconnected to each other in a daisy- chained fashion and four sensor pairs each containing two pairs of infrared transmitters and receivers. These modules are,

1. Odometer / Power Module (OPM)
2. Passenger Count Module (PCM)

3. Data Storage Module (DSM)
4. Diagnostic Module (DGM)

A typical VDAS will comprise of one OPM, one DSM and two PCMs, one each for front and rear doors. A Diagnostic Module (DGM) can be connected to the system in order to examine certain operational and functional parameters of the system. The functions of each of these modules are described below.

A.1.1 Odometer / Power Module

This module is fed directly from a 12 volt battery and rest of the modules in VDAS get their power from this module via interconnecting wires. Apart from serving as power distribution center for the VDAS system OPM also accepts pulses from the odometer as its inputs. The incoming pulse train is divided and then counted to get a measure of distance travelled by the vehicle. The division factor can be chosen to be anywhere from 1 to 4095 by installing the proper jumpers. Presently it is set up for divide by one operation. The vehicle movement information is passed on to other modules so that appropriate logs can be generated.

A.1.2 Passenger Count Module

Two PCMs are used in each VDAS system, one for the front door and another for the rear door monitoring. Both of these are connected with remaining two modules in a daisy-chain fashion. Each module monitors the status of doors via an ON/OFF contact signal and passenger activity via infrared sensor pairs mounted on the inner walls of the passage of each door. Each sensor set consists of two pairs of emitters and detectors generating four infrared beams. The order in which these beams are broken and restored, due to passenger movements, determines the number of passengers boarding and alighting from that door. Door open/close signal and vehicle movement information is used by these modules to generate appropriate logs.

A.1.3 Data Storage Module

The primary function of this module is to receive and retain the logs generated by remaining modules and store them in the order of their arrival which is same as the order of their generation. The other modules in system use special flags to signal a particular type of event to DSM so that DSM can initiate requisite data transfer from that module to prepare a log. The data stored in the DSM can be transferred to a computer via a serial link for further processing and analysis.

A.1.4 Diagnostic Module

This module provides diagnostic facilities for all other modules in a VDAS. It is a portable module with a 16-key keypad, 80 character alphanumeric LCD display and serial communications capabilities. Diagnostic functions are organized hierarchically in a tree structure form. On power up or by pressing key 'E' it resets to Level 0 (root). The next level is entered by choosing the module identifier keys. Access to diagnostics specific to a module or to module data bus can be invoked by a single key. Once a particular module has been selected then various diagnostic functions pertaining to that module or data bus can be selected by single keys. A menu helps user identify the function and its corresponding key.

A.2 Sensors

Each system requires four pairs of sensor units and each unit consist of two pairs of infrared transmitters and receivers. Two pairs of such sensor units are mounted in the door-well of both the front and rear doors. These sensor units are connected to Passenger Count Modules and get the power from them. In power-up condition each transmitter produces a beam which is received by a corresponding receiver. There are four such beams for each door. The broken and unbroken states of these beams in conjunction with the order in which they were broken and made is used by the system to arrive at the conclusion regarding the passenger activity.

A.3 Communications

This VDAS system provides two different types of communications method, one for inter-module communication and another for downloading of accumulated data to a computer. Both of these are achieved by using a packet based approach for requests, acknowledgements and actual data transfers. The communications with the outside world is by means of a RS-232 interface without any handshakes. These two methods as employed in Red-Pine's VDAS system have been discussed below.

A.3.1 Inter-module Communication

All data transfers between modules takes place by means of exchange of a specific structure called packet. Each packet is bounded by a unique double byte header and a unique double byte trailer and and these two patterns are not permitted to be in the information field of the packets.

The structure of a typical packet is as shown in Figure 34 on page 126. The format of the packet is,

Header	ESC STX, a double byte sequence (1BH 02H)
Length	Number of bytes to follow in the range of 1 to 255.
Information	Data bytes indicated in the length field.
Checksum	Double byte, 2's compliment of 16 bit sum of length byte and information bytes.
Trailer	ESC ETX, a double byte sequence (1BH 03H)

In order to preserve the uniqueness of header and trailer double bytes the occurrence of ESC (1BH) in any byte not belonging to header or trailer is replaced by another unique double byte sequence of ESC DLE (1BH 10H). This scheme prevents the generation of header or trailer patterns within the data packet.

A.3.2 Communication for Downloading

The VDAS responds to specific commands sent from a computer outside it. The format of communication is a variation of packet based system. The computer has to send a specific type of packet to request a particular type of service or action by VDAS. The request packets are five byte long. The format of the request packet is given below.

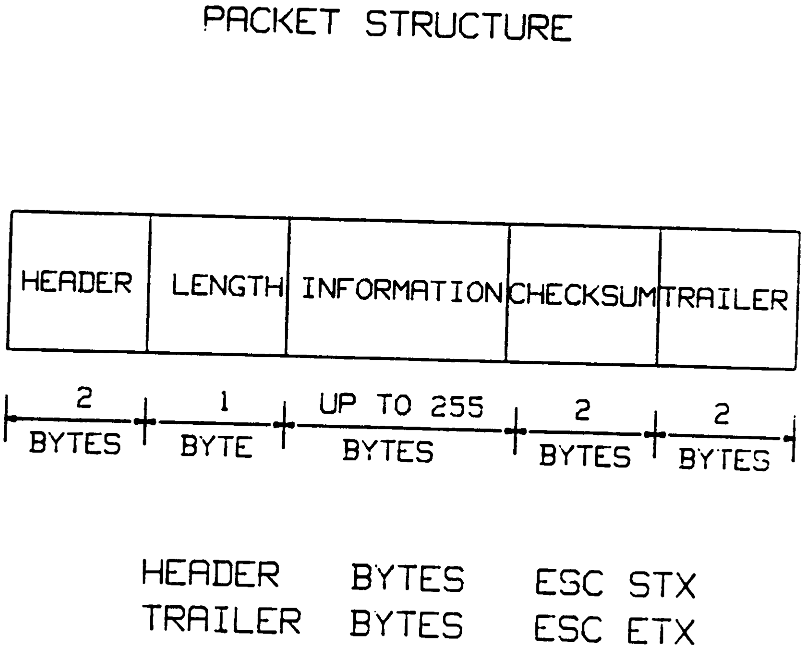


Figure 34. The Structure of Packets used for Communication

First Byte	FF Hex
Second Byte	FF Hex
Third Byte	Upper Byte of Address (Normally F0 Hex).
Fourth Byte	Lower Byte of Address (Normally 00, 01 or 02).
Fifth Byte	Checksum of Byte 3 and Byte 4.

The format of response packets generated by DSM is given below.

First Byte	FF Hex
Second Byte	FF Hex
Third Byte	Length Byte - Indicates the number of bytes following.
Fourth Byte	data byte
Fifth Byte	data byte
	.
	.
Second Last	data byte
Last Byte	Checksum of length byte and all data bytes.

As is evident from the above each packet has a two byte header which is always same (FF FF) and one checksum byte as trailer. The third and fourth bytes carry the commands to DSM. These are,

F0 00	Begin Dump command.
F0 01	Last Log Address command.
F0 02	Reset DSM Logs command.

In response to "Begin Dump" command DSM generates logs for downloading and acknowledges the receipt of command by sending a one byte response of DD Hex in its reply packet. The requesting computer has to keep sending request packets with "F0 00" command till DSM returns an acknowledgement byte of A1 Hex. After this it will return logs in blocks of 128 bytes.

In response to "Last Log Address" command signified by "FF 01" the DSM returns a six byte reply packet with the two byte long address of last log sandwiched between length and checksum bytes.

In response to "Reset DSM Logs" command indicated by "FF 02" the DSM returns a reply packet with one byte response of A1 Hex.

A.4 Count Generation Algorithm

There are four sensor pairs for each door generating four beams. In normal powered up state the beams between each of sensor pairs are made. They are broken when an opaque object crosses the beam that is gets in between the transmitter and receiver. The order in which beams are made and broken determines whether a passenger is climbing aboard are alighting.

This system requires that all four beams in a doorwell are to broken first before any activity is registered. Let the beam near the door (corresponding to outer sensor) is labeled 1 and the beams farthest from the door is labeled 4.
Please see Figure 35 on page 130for sensor positioning and beam labeling. If all four beams are broken and then made in order 1, 2, 3 and 4 an on count is generated. If all four beams are broken in order 4, 3, 2 and 1 and then made then an off count is registered. It was noted that an off count is generated whenever beams are made in reverse order that is 4, 3, 2 and 1 irrespective of the order in which they were broken.

This can be summarized as shown below.

1. On Count Generation

   - B1, B2, B3, B4 AND M1, M2, M3, M4.

   - B1 followed by breaking of remaining three and then all made.

2. Off Count Generation

   - B4, B3, B2, B1 AND M1, M2, M3, M4

   - B4 followed by breaking of remaining three and then all made.

A.5 Log Structure

This section describes various types of logs generated by different modules in a typical VDAS system. For each type of log following information is provided.

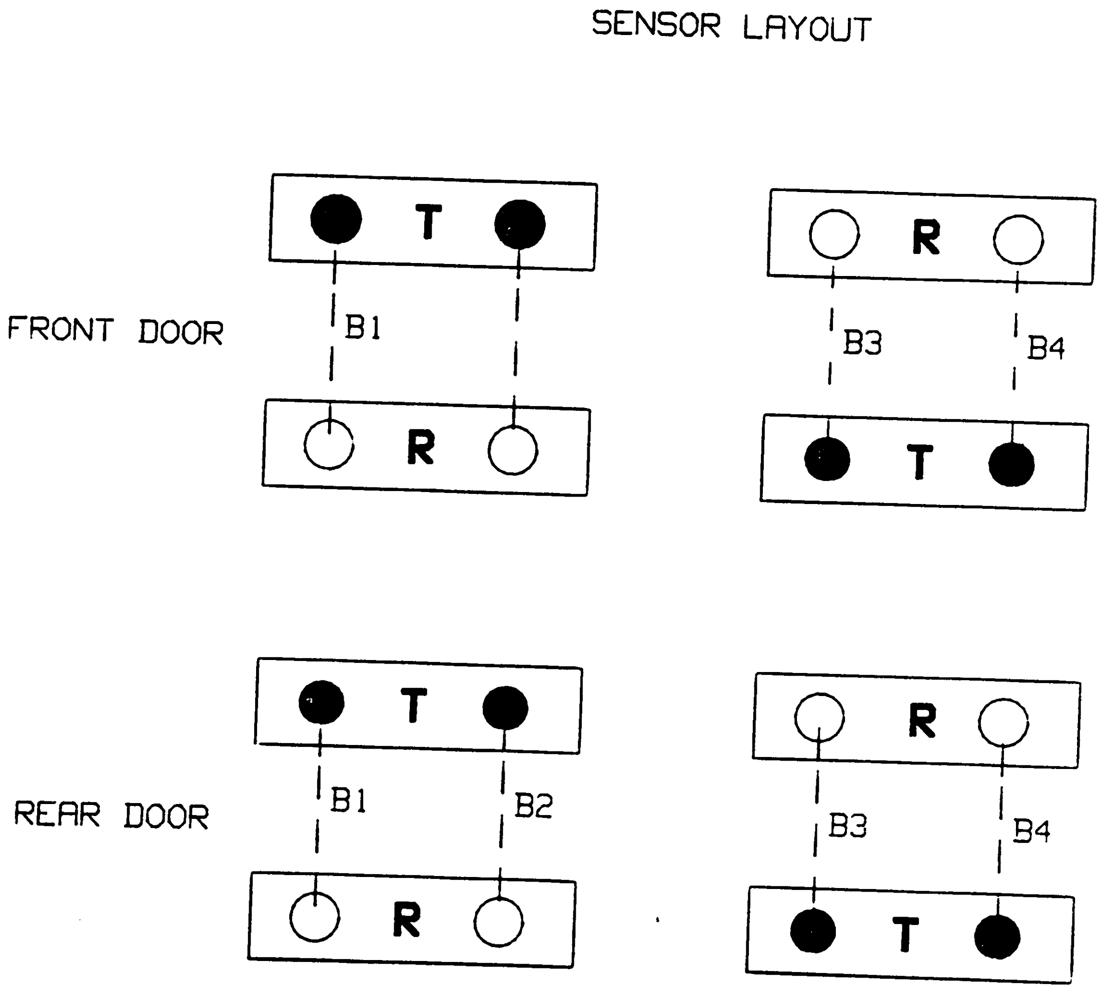


Figure 35. Red Pine System Sensor Arrangement

1. The type of log and its name if any.

2. The information contained in the log and its function.

3. Source and destination of the log.

In the present system there are in all 15 different type of logs numbered from 0 to 14 (0 t0 0E Hex). Each of these logs are associated with a different type of event [31].

Log Type 1	Power up log, is the very first log generated after power on.
Log Type 2	Hour overflow log. It is generated when an hour expires. Since time is kept in 15 sec ticks one byte can hold only enough to make one hour and it will overflow beyond that.
Log Type 3	One minute idle log - generated when vehicle remains stationery for a period of a minute.
Log Type 4	Distance overflow log - generated when distance, counted as odometer transitions, exceeds the value that can be held in a byte.
Log Type 5	Passenger log - generated after vehicle moves and if there was passenger activity at the previous stop.
Log Type 6	End of two or more minute idle log - generated when vehicle moves after being stationery for a period of more than two minutes.
Log Type 7	Dump forced elapsed time log - generated when a dump is initiated and contains 24 bit time value representing the time expired since last dump or power up.

Log Type 8	Dump forced total distance - generated when a dump is initiated and contains 24 bit value representing total odometer transition count since last dump or power on.
Log Type 9	Front Passenger Count Module not responding log.
Log Type 10	Rear Passenger Count Module not responding log.
Log Type 11	External Input (signpost) log - generated whenever vehicle detects a signpost by means of an infrared device or by operator action of changing the board.
Log Type 12	Distance overflow overrun log.
Log Type 13	Odometer Power Module not responding.
Log Type 14	Vehicle Identification log - contains 8 digit vehicle ID.

A.6 Test Procedure

A vehicular data acquisition system was assembled using the modules mentioned above. The interconnections were made as described in the manual supplied by Red Pine Associates. A schematic of interconnections is shown in Figure 36 on page 133.

The two sensor pairs each were connected to front and rear doors respectively. The door ON/OFF signals for both the doors were generated by an on/off switch. The odometer pulses were simulated by a push button switch (a signal generator can be used).

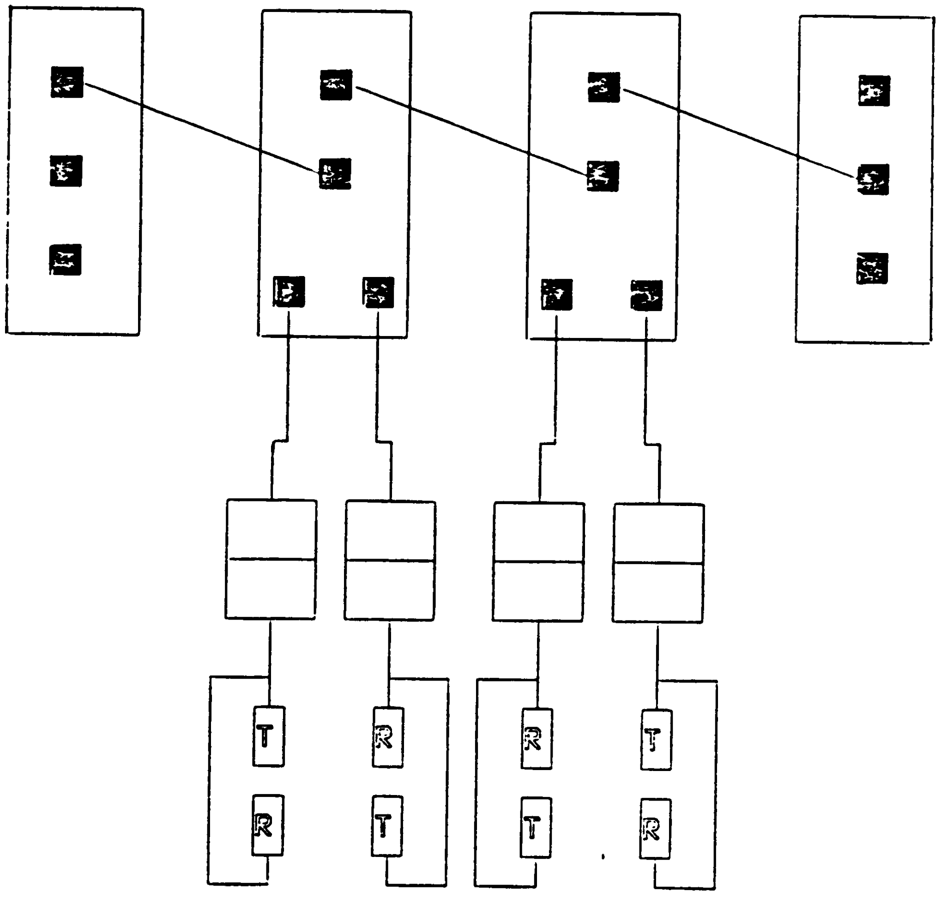


Figure 36. Red Pine System Interconnections

Next step is to generate all possible types of logs by simulating passenger boarding and alighting activities, door open and close signals, distance measurements by counting pulses from signal generator. The system does time measurements by maintaining an interrupt driven timer. These simulations are done as discussed below.

- Passenger boarding activities are simulated by passing an opaque piece of card board between the sensors such that the sensor set labeled **outer** is encountered first followed by sensor set labeled **inner** Each such movement simulates boarding of one passenger and results in increment of previous **on count** value.

- Passenger alighting activity is simulated exactly as above except this time inner sensor set gets affected first. Each such movement indicates a passenger disembarking the vehicle and thus incrementing the previous **off count** value.

- Before passenger movement is simulated the door open/close signal should be set to **open** for both the doors. It is done by means of an on/off switch, the 'on' position indicating open and 'off' indicating closed.

- Vehicle movement is simulated by feeding certain number of pulses to its odometer input using a switch and or a signal generator.

- Module-not-responding mode is simulated by unplugging that module from the system. However it is applicable only to PCM and DSM since power is cutoff to the system if OPM is removed.

Now a VDAS is assembled using the modules described above and it is connected to the "Simulation Panel" which contains switches, and mounted sensor pairs, a switch and terminal for odometer input and a connector for Diagnostic Module. This system is comprehensively tested by simulating various possible conditions of passenger movement and vehicle travel. The activities of the system are observed through the Diagnostic Module by selecting different modules and different levels. These observations are then compared with the documented results for these situations as specified in the manual for this system.

Availability of one perfectly working VDAS system is the precondition to test individual modules since no test instructions are available for these modules. Moreover due to lack of any hardware and more importantly software documentation our understanding of these module is limited to functional level only. The approach taken here is to remove one single module from the working VDAS system and replace it with the module-under-test. Now the same simulation steps are followed which were used to establish the integrity of original system. If the system continues to function in the specified fashion it can be safely concluded that module-under-test is functionally alright.

A.7 Observations

The following observations were made while a VDAS system was assembled and made operational.

1. The complete system when operational requires about 1.7 Amp of current at 12 volts.

2. For proper count generation all four beams must have been in broken state at some point of time before they are made.

3. Occasionally the counts logged in DSM are different than recorded in PCMs as observed via Diagnostic unit. It was noted that both on and off counts were logged incorrectly.

4. Diagnostic Unit continues to scan recorded logs in "DSM Logs" modes beyond current log displaying illegal log types and no error indication.

5. The DSM accepts a true RS-232 compatible signal at its receive line but generates only a TTL signal at its transmit line. Consequently a line driver chip, MC 1488, was used to convert this TTL signal into a RS-232 compatible signal required by IBM PC.

6. The DSM requires two or more transmissions of the request packet before it is able to transmit data for downloading. In response to first request it produces a 5 byte packet containing a one byte message DD.

7. In response to various requests denoted by Hi and Lo byte addresses of 'FF 00', 'FF 01' and 'FF 02' it transmits a large number of bytes instead of 5 in two cases and 128 in another. The DGM display however does not show it but it can be easily verified by an oscilloscope.

8. Even though the DSM transmits packets which are 5 or more bytes long the Diagnostic Unit does not display actual packet transmitted instead it removes the two byte header 'FF FF' and the checksum byte. However the receive packet is displayed in its entirety.

9. There is no self test mode hence it is difficult to be sure whether system is functioning properly. There are however LEDs mounted on the circuit boards which pulse at about 1 Hz to indicate module healthy state. But they are inside the encapsulated boards and not visible from outside.

10. The connector wiring to Odometer Power Module was modified to make it consistent with the documentation.

A.8 Results

One complete VDAS system was assembled and its operation was verified by simulating various events and examining its response. Extensive testing was done on this system to make sure that it successfully responds to all possible situations and produces appropriate logs. Subsequently the remaining two sets of modules were tested in the manner described above and were found to be functional.

Appendix B. MSI-C988 Microcomputer Card

The MSI-C988 is a Multifunction Microcomputer card, which incorporates a CMOS 80C88 microprocessor and operates from a single +5V supply. This Microcomputer card is well suited for real time monitoring and control applications requiring minimal hardware and low power operation.

The card provides 64K bytes of on-board memory, 32K bytes each of RAM (with battery back up) and PROM, a real time clock with battery back up, two RS-232C serial I/O ports, 24 parallel I/O lines, three 16-bit programmable counter/timers, a 10-bit analog input and a cascadable interrupt controller. It uses clock frequency of 5MHz.
Please see Figure 37 on page 139 for the functional block diagram. The salient features of this board have been summarized below.

- 80C88 CMOS Microprocessor

- Up to 65,536 Bytes Onboard RAM/PROM

- 58167 Real Time Clock with Calendar

- Two RS-232C Serial Ports from 110 to 38,400 BAUD

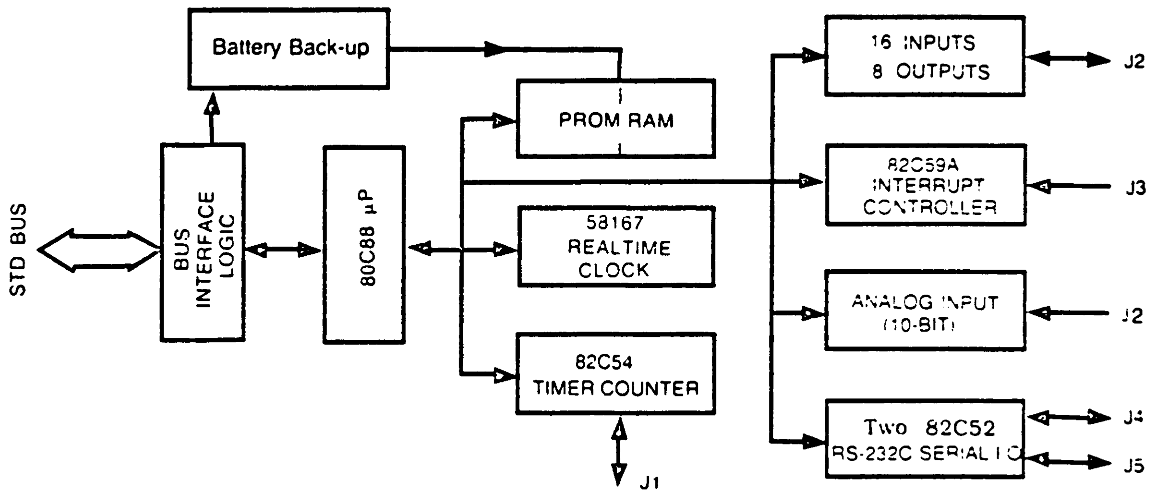


Figure 37. Block Diagram of C-988 Microcomputer Board.

- 24 Parallel I/O Lines (16 In/8 Out)
- Three 16-bit Programmable Timers / Counters
- A 10-bit Analog Input
- 82C59A Cascadable Interrupt Controller
- Onboard Execution Monitor PROM for IBM PC/XT/AT
- Power requirement 5V ±5 percent, 50 mA typical

The 64K bytes of memory enables memory-mapping applications also. It provides two sockets for accommodating PROM and RAM memory using either 27C64 or 27C256 type PROM devices and 5564 (8K bytes) or 55257 (32K bytes) type RAM. The two sockets can be configured in a variety of ways for these memory devices.
The valid address regions for the on-board memory are shown in Figure 38 on page 141.

B.1 Input Output Devices

The C-988 microcomputer board contains two CMOS Serial Controller Interface devices (82C52) providing two RS-232C channels, a CMOS Programmable Interval Timer (82C54), a CMOS Programmable Interrupt Controller (82C59A), a Real Time Clock (NS 58167), an Analog to Digital Converter (ADC 1005) for analog input and buffers (74HC244) for parallel inputs and outputs.

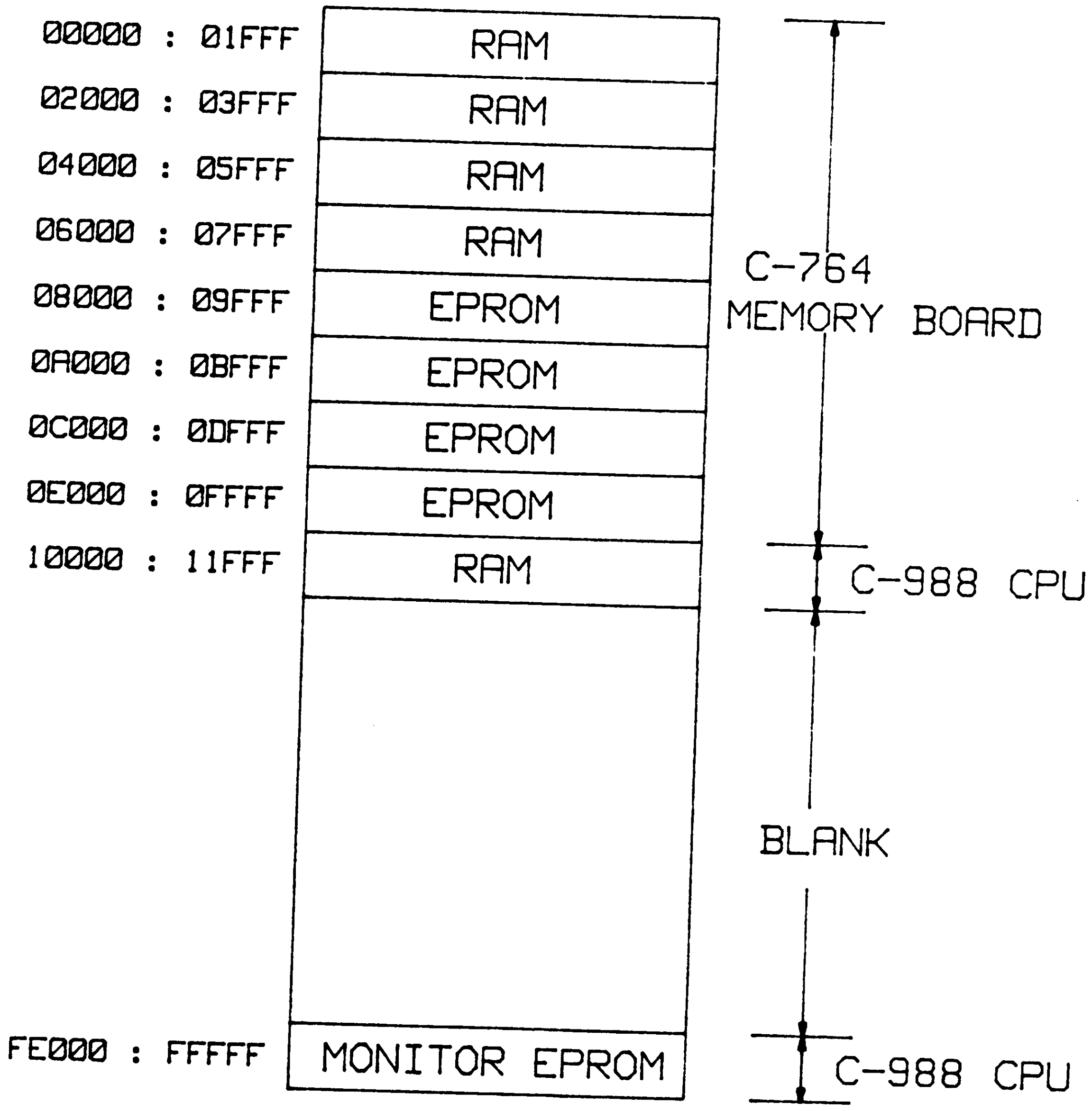


Figure 38. Memory Address Map

Table 1. I/O Map for MSI-C988 Microcomputer

I/O Device	Register	Address
USART 1 82C52	Usart Control Register	01H
	Modem Control Register	02H
	Baud Select Register	03H
	Data Register	00H
USART 2 82C52	Usart Control Register	21H
	Modem Control Register	22H
	Baud Select Register	23H
	Data Register	20H
PIT 82C54	Control Register	63H
	Counter 0	60H
	Counter 1	61H
	Counter 2	62H
PIC 82C59A	Control Port	40H
	Data Port	41H
RTC	Counter(1/10000)	C0H
	Counter(1/10,1/100)	C1H
	Counter(sec)	C2H
	Counter(min)	C3H
	Counter(hour)	C4H
	Counter(day of wk)	C5H
	Counter(day of mo)	C6H
	RAM(1/10000)	C7H
	RAM(1/10,1/100)	C8H
	RAM(sec)	C9H
	RAM(min)	CAH
	RAM(hour)	CBH
	RAM(day of wk)	CCH
	RAM(day of mo)	CDH
	Interrupt Status	D0H
	Interrupt Control	D1H
	Reset Counters	D2H
	Reset RAM	D3H
	Status Bit	D4H
	GO Command	D5H
	Standby Interrupt	D6H
	Test Mode	DFH

The detailed function and programming information on 82C52, 82C54, 82C59A can be found in Harris CMOS Data Book [10] and also in MSI-C988 Users Manual [24]. The Table 1 on page 142 contains the addresses of I/O devices installed on C-988 board.

B.2 MSI-C988 Monitor

A monitor program is supplied with the MSI-C988 microcomputer which provides a low level interface between the user and the hardware. The monitor requires that a CRT or an IBM PC/XT/AT personal computer connected to RS-232C serial port at connector J4. The required baud rate is 4800. It is supplied on a 8K X 8 PROM and provides twelve commands for exercising scratchpad registers, memory and I/O ports, as well as permitting file transfers to and from IBM PC [26].

The monitor requirements for useful operation are listed below.

- The 8K PROM must be located from 0FE000 - 0FFFFFH which corresponds to socket U16 on C-988 board.

- The monitor needs about 700 bytes of RAM for its own use and expects it to be located from 00000 onwards. This corresponds to RAM in socket U15 on C-988 board if it is jumpered as on board memory (MEMEX = 0). Otherwise it corresponds to socket U8 on C-764 Memory Board which must have RAM.

- It uses interrupt lines IR1 and IR2 on 82C59A interrupt controller for receive and transmit operations. The controller is initialized so that these correspond to interrupt type 9 and type 10 (INT 9 and INT 10).

- It also uses interrupt type 3 (INT 3) for breakpoint interrupts.

- It uses serial port (J4) to communicate to a CRT or an IBM PC at a baud rate of 4800. The interrupts used are INT 9 and INT 10 with handlers located at 024H and 028H for receive and transmit interrupts respectively.

B.3 Command Summary

The commands provided by the monitor has been listed below with the function they perform along with the command formats. Note that all numbers are expected in **hexadecimal only**, all addresses are to be specified in the form of **SEG:OFFSET** and all commands must be entered in capitals only. For further details on these commands the reader is referred to [26].

B.3.1 Arithmetic (Hex) Command - A

It performs the hexadecimal arithmetic sum and difference of two operands. The A command, entered as A <P>,<Q>

B.3.2 Change Memory Command - C

The change memory command is entered as

C<address>, <old value>-<new value> where <address> is the memory address and <newvalue> is the value that replaces <old value> if entered.

B.3.3 Display memory command - D

The D command is entered as D <lo address>,<hi address>

It displays memory beginning at <lo address> and ending at <hi address> to be displayed on the console device.

B.3.4 Examine Register command - E

The register command, entered as E or E<register> is used to display and change the 80C88 registers.

B.3.5 GO command - G

The G command is entered as G <start address>,<breakpoint 1>,<breakpoint 2>

It permits execution of a user program under monitor control. The first parameter <start address> is the starting address of program execution. If specified, program execution will begin at this address. If not, execution will begin at present value of program counter. If either break point is specified (<breakpoint 1>, <breakpoint 2>), execution will terminate when this address is encountered and the monitor will echo the value of the program counter.

B.3.6 HELP command - H

The H command is executed by simply entering H. A summary of the monitor commands are listed on console device.

B.3.7 INPUT command - I

The input command is entered as I<port> where <port> is the desired port. Initialization required for the input port must be performed by the user prior to the command.

B.3.8 OUTPUT command - O

The output command is of the form O <port>,<value> The command outputs to <port> the Byte <value>. Appropriate initializations must be performed by the user prior to using the command.

B.3.9 READ command - R

The read command is executed by R <start address>

It is used to read a file on the IBM PC into memory via the console serial input port. The read command must be given in conjunction with a write file command on the IBM terminal handler program. The read command is executed first and then the write file command on terminal handler.

B.3.10 SET memory command - S

The set memory command as shown. S <lo address>,<hi address>,<value>

It is used to set memory from <lo address> to <hi address> with the constant <value>

B.3.11 Transfer memory command - T

The transfer memory command is, T <lo address>,<hi address>,<new address>

It is used to transfer the memory from <lo address> to <hi address> into a memory block beginning at <new address>.

B.3.12 WRITE command - W

The write command is executed by W <start address>,<# byte>

It is used to write <# bytes> from <lo address> to a file on the IBM PC.The write command is used in conjunction with the read file. The read command is executed first and then the write.

B.4 Terminal Handler Program

A terminal handler program called TERM.EXE is supplied on a diskette to be used in conjunction with C988 monitor for communicating with the IBM PC. This program converts the IBM PC into a dumb terminal. It requires that serial port J4 of microcomputer board is connected to COM1 port of the PC and must operate at 4800 baud. Apart from echoing the commands on the screen it also provides two very important facilities for file transfer to and from the PC and the C988 board. These commands called READ FILE and WRITE FILE are used together with READ and WRITE commands of monitor. They are invoked by typing Control-E on the PC keyboard. The program will then request the file name which will be used for reading or writing.

For reading a file from C988 into PC first type W with its parameters and then type control-E. Enter 1 for read on PC and then enter file name in which it is to be written. Similar procedure is used for downloading a file from the PC to target board which starts with a R command. Please see MSI-C988 Monitor [26] for further details.

B.5 MSI-C764 Memory Card

This card works on CMOS STD BUS standard and works with CMOS and NMOS processor cards based on Z80, 8085, 8088, NSC800 and 8080A microprocessors [25]. It has a combination of CMOS PROM/RAM with battery back up capability for RAM. There is a provision for 64K bytes of 2764 type PROM and RAM.

The card will accommodate either 8K bytes of 2716/6116 PROM/RAM, 64K bytes of 27C64/5564 PROM/RAM, or 32K bytes of 27C32 PROM. NMOS types 2716, 2732 or 2764 can also be used. The salient features are listed below.

- 65,536 Bytes 27C64/5564 PROM/RAM
- 32,768 Bytes 27C32 PROM
- 16,384 Bytes 2716/6116 PROM/RAM
- RAM Battery Back up (MSI-C764A)
- MEMEX Decoding for up to 128K Bytes in the System
- Universal Processor compatibility
- CMOS Components for Low power operation and improved noise immunity
- Industrial Temperature Operating Range
- Vcc= 5v $\pm$5 percent tolerance at 50 micro ampere typical with memory sockets empty.

Appendix C. Overview of Software Development

The software development process for embedded microprocessor systems is different from conventional software development methods. Both of these methods involve the cycles of edit, compile/assemble, link, load and test programs but similarity ends there. The fundamental difference between the conventional method and embedded microprocessor method is the development environment.

In conventional method the computer itself provides a software development environment in the sense that all the utilities like compilers, linkers etc. reside in the computer and the executable module generated after link and load operation executes on the same computer. Since most of such computers have an operating system providing a higher level of interaction with hardware the application programs use the services provided by it than manipulating the hardware.

The scene is exactly the opposite in case of embedded microprocessor systems. Normally they do not contain any operating system and provide primitive services like downloading etc. via a monitor program. Since they do not provide utilities for editing, assembling or compiling, linking etc. there is a need for another system which provides these. These systems are called Microcomputer Development Systems. A detailed discussion of microcomputer development systems have been provided in [37]. The development computer is used for writing, translating and linking program modules while the target computer is used for testing these modules [8]. The embedded microprocessor in the target computer executes the programs developed on a different computer for it. It may contain VRTX/OS that is VRTX alone or along with one or both IOX and FMX.

The desirable features of a development computer are completely different. They normally support a sophisticated operating system like MSDOS or ISIS along with a good text editor. Cross-assemblers, cross-compilers and cross-linkers are other critical tools provided by development computers. The cross-compilers and cross-assemblers run on the development machine but produce code which is executable on the target machine. The cross-linker combines separately assembled program modules into one loadable and executable module. Some linkers however would generate only relocatable code for example, Microsoft Linker. The linked module has to be located to run at a physical address. This action can be performed by a loader or a locater program.

In case the program modules are developed in a higher level language then a cross-compiler will be used to compile these into object modules which in turn will be linked with each other and with interface libraries to produce absolute code. The located output is transferred to target computer's RAM via a download link. The downloading operation can either be performed by the emulator or by the monitor on the target machine. Figure 39 on page 152 shows a typical development computer with the software tools supported and its relationship with target computer.

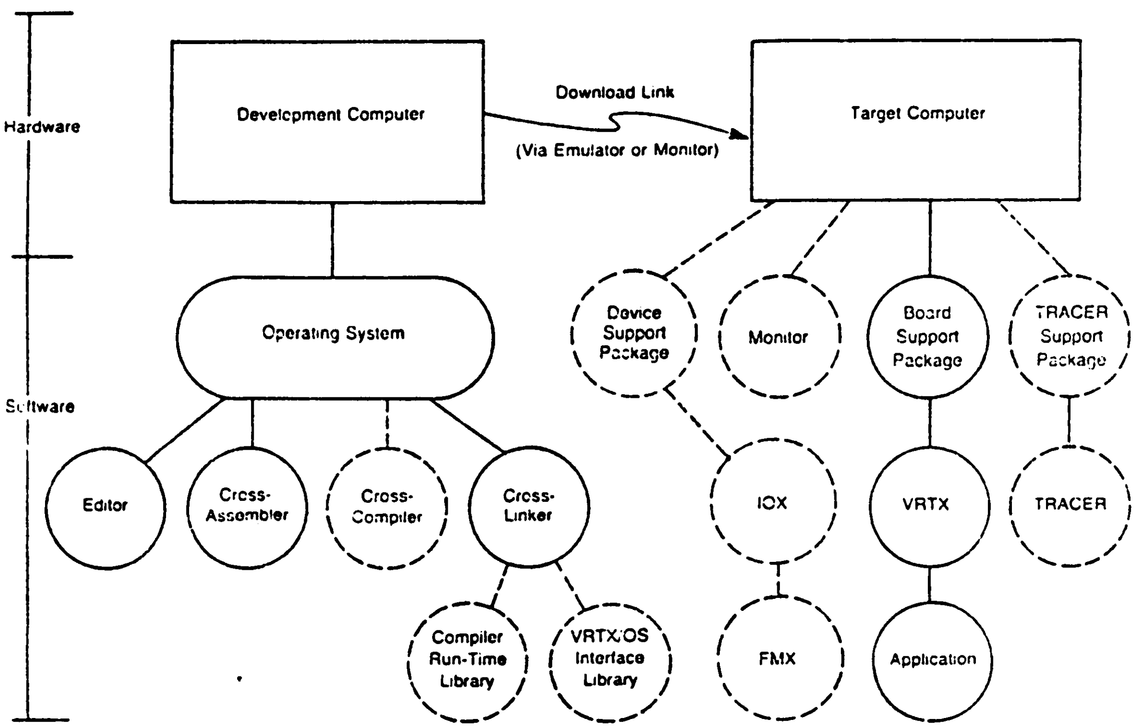


Figure 39. Software Development Environment for Embedded Systems [8]

C.1 Software Development Cycle

The software development for an embedded microprocessor system is a cyclical and incremental process as shown in Figure 40 on page 154. The steps in a typical cycle are listed below [8].

- Write program modules using the editor on the development computer.
- Compile and or assemble each module into object modules.
- Link the object modules together along with the language interface and run time libraries as needed.
- Create an absolute module ready for loading and execution using the loader or locater utility on the development computer.
- Download the absolute module into the target computer's RAM.
- Execute this program on target computer and determine any errant behavior. Process complete if everything is as expected.
- Modify the program and repeat from the beginning.

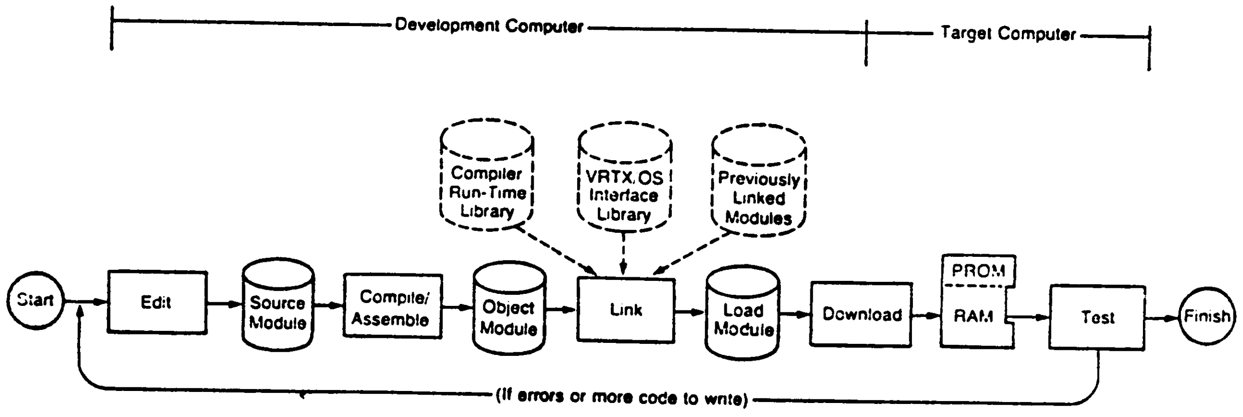


Figure 40. Software Development Cycle for Embedded Systems [8]

C.2 Using IBM PC as Development Computer

It was decided early on to use an IBM PC as a development computer to do entire software development for porting of VRTX and as well as the application. IBM PC can easily be used as a development station for generating executable code for target machine provided it contains an 8088 or 8086 processor. In case the target system has another type of processor then cross-assembler, cross-compiler and cross-linker tools would be required.

The IBM PC used for this project supports MSDOS version 3.1 operating system which provides a host of file management services. Also there is a choice of line and screen editors available which can be used for creating and modifying program modules. Microsoft Macro Assembler version 4.0 and Microsoft Linker 3.55 are available on this system and were used for assembling and linking various program modules. The support for Microsoft C version 4.0 is also available.

The Microsoft Linker (LINK) generates a relocatable file in the ".EXE" format of MSDOS and it is the one which locates this linked module at a physical address to be executed inside PC's memory. This file can not be used directly for execution on a target system. Another utility called PCLOCATE version 2.1 from ALDIA Systems was used to convert this relocatable ".EXE" file into an absolute file with physical addresses incorporated. PCLOCATE is a very helpful tool in the sense that it not only generates an absolute file for downloading onto a target board but also generates files with even and odd bytes in Intel hex format for burning the EPROMs on standard PROM programmers.

However the file of immediate concern in this case was of type ".TMP" generated by PCLOCATE which can now be downloaded onto the target board using the onboard monitor. This requires that the locate addresses must match the downloading addresses specified while invoking the monitor command. This program has absolute addresses and hence can be executed on the target system.

The development cycle is given below.

- Create or modify program modules using an editor like Wordstar or Dved.
- Use Microsoft's Macro-assembler (MASM) to assemble various assembly language source modules into object modules.
- Use Microsoft's C compiler to compile various modules written in C
- Use Microsoft's linker (LINK) with /map switch to link all of these object modules, language reference libraries and run time libraries.
- Use ALDIA Systems' PCLOCATE to generate an absolute code module. Note that the start addresses for various segments specified while using this utility must physically be available. Apart from segment start addresses file start and stop addresses are also specified.
- Connect COM1 port to target computer serial port (J4) and execute TERM.EXE program on C-988 monitor diskette. Reset the target system to observe signon message confirming proper connection.
- Use monitor's Read (R) command with the address same as specified as start address while locating.
- Type Control-E and reply 2 to question indicating write operation.
- Type in the file name and extension with drive designator if not on same drive followed by a carriage return.

- A star "*" prompt indicates proper downloading else an error message "ERROR" will appear. Repeat previous three steps in case of error.

- Execute this program using monitor's Go (G) command. Note the start of execution address does not necessarily have to be the start address specified previously.

- Observe the program behavior. Repeat the process from the beginning if program does not perform intended function.

Please refer to Microsoft's Macro Assembler Manual [22] and IBM PC DOS 3.1 Manual [16] for any assembly error, link errors and COM port details. Details of ASM-86 assembly language can be found in [13] and [14]. The information about PCLOCATE can be found in [28].

Appendix D. Task Description

This section describes some of the various tasks that are required to implement the functions of a vehicular data acquisition system. The discussion includes almost all of the critical tasks and traces their design and functioning. Please note that this discussion attempts to describe the functioning of a task not actual coding. Hence reader is advised against trying to find one to one corresponds between the description here and the actual implementation. There are certain situations where it was determined that the signalling mechanism has to be modified to suit the purpose instead of using a standard system call. Though no explicit system calls were added that effect was obtained by writing a routine that is not part of VRTX but performs a similar function.

D.1 Vehicle Time Management Tasks.

The tasks under this category are designed for management of basic timing requirements.

D.1.1 System Timer Task

This is one of the highest priority tasks in the system and its sole purpose is to keep VRTX posted on expiry of each timer tick. Generally an external source or self loading timing device like 8253 PIT is used to generate continuous ticks at a desired frequency. In our case an 82C54 PIT provides 10 millisecond ticks. Each tick interrupts the processor and its service routine issues a VRTX timer call UI_TIMER to inform it that a timer tick just expired. The purpose of the system timer is to provide a free running clock to the application software which will use it to time stamp various events.

D.1.2 Elapsed Time Task

This task maintains a count of time ticks since the last time logs were dumped or power on. The purpose of this task is to determine the total amount of time that elapsed since the previous dump. This information is used when the logs are dumped and another log will be created containing this data.

D.1.3 Elapsed Distance Task

This task maintains a count of odometer transitions since the last time logs were dumped or power on. The purpose of this task is to determine the total distance that was covered since the previous dump. This information is used when the logs are dumped and another log will be created containing this data.

D.2 Power On Log Generation Task

This task is scheduled to run whenever system is powered up. The power on situation must be detected by some means to activate this task. If the system is so designed that a power on interrupt is generated then the service routine of that interrupt can activate this task. Else this is to be executed explicitly as the very first task. The purpose of this task is to create a power on log containing the time. This is achieved by making this task obtain the time information and then calling the subroutine log_write which will actually store this log inside the memory. After the log is stored this task is never going to run till next power on hence it can be easily deleted.

D.3 Vehicle Idle Control Tasks

The tasks under this group are meant for monitoring and recording various idling periods as required by specifications.

D.3.1 One Minute Idle Task

The function of this task is to monitor the duration for which the vehicle is stationary and generate sufficient data so that a log indicating this situation can be created. As is obvious from the requirements of this task it needs both distance and time information to do the job. In order to determine whether or not the vehicle is stationary it can look at the odometer transitions. The service routine of odometer interrupt must post the arrival of transitions to a mailbox which can then be tested by this task to gain information about vehicle movement.

This task has nothing to do as long as the vehicle is in motion but as soon as it detects that vehicle is not moving it will start monitoring the duration and will continue to do so till either the duration exceeds one minute or the vehicle moves. Note the movement of the vehicle results in suspension of this task. In case the vehicle moves within a minute no action is taken and task gets suspended again. But if duration exceeded one minute then the log writing tasks at lower level need be informed about this situation which is accomplished by posting a message to the task responsible for creating this log.

D.3.2 Two or More Minute Idle Task

This task is similar to the one described above. Apart from monitoring the movement of the vehicle it also monitors the entire duration of the halt. This is achieved by obtaining the VRTX time in terms of clock ticks by using sc_gtime call. The tasks pends for two minutes at a mailbox which is posted to by odometer interrupt handler. In case the vehicle was stationary for a period exceeding two minutes this will pend again till the vehicle moves and odometer handler gets a chance to post a message. At this point new value of VRTX time is obtained to determine the entire duration of halt. This information is then passed to a task which is responsible for writing this type of log.

D.4 Overflow Control Tasks

The tasks in this category are designed to address the situations in which the value of a parameter being accumulated exceeds the storage allocated to it.

D.4.1 Distance Overflow Task

This task keeps monitoring the distance covered by the vehicle since the last log was generated. Since most logs must contain the information about the distance covered since last stop and there is only one byte available in a log to hold this data it is necessary to make sure that overflow is detected.

This task monitors the odometer pulses via a mailbox posted to by the odometer interrupt handler and maintains an internal distance counter. This counter is reset whenever a log is written since the log writing tasks post a message to this task informing it to restart. In case the vehicle moves long enough that the distance counter can no longer hold the value a message is posted to another task which generates a distance overflow log.

Please see Figure 41 on page 164 for sequence of events.

D.4.2 Time Overflow Task

This task is very similar to the one described right above. The difference between the two is that this one keeps time instead of keeping distance. This time corresponds to duration for which vehicle has been in motion. The time is maintained as the current value of real time in minutes and seconds. It requires two bytes in the log to hold the current value of time. In order to take care of overflow problem an hour overflow log is generated every hour.

Please see Figure 42 on page 165.

D.5 Count Generation Tasks

The tasks in this category handle situations that may result in counts. They also interact with sensors via an ISR.

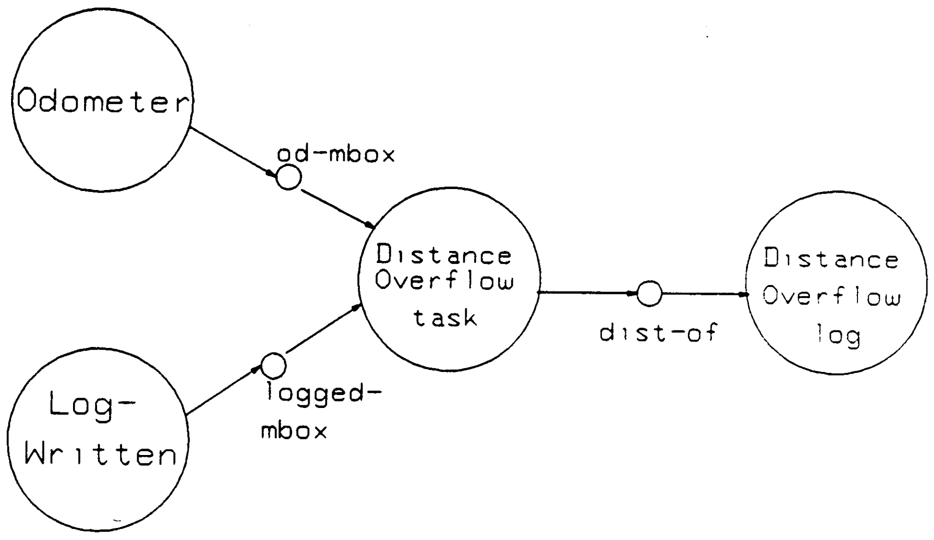


Figure 41. Distance Overflow

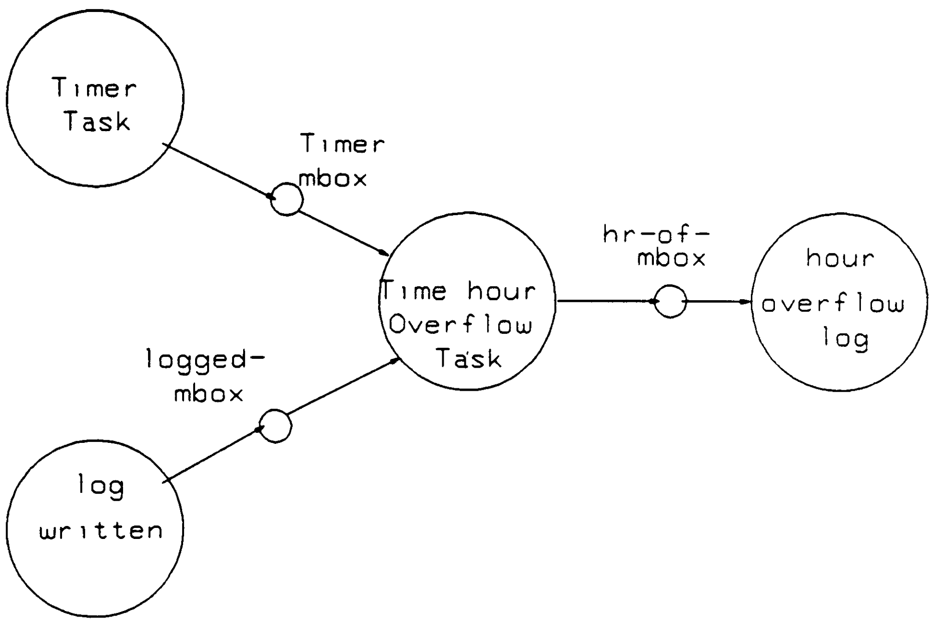


Figure 42. Hour Overflow

D.5.1 Break_on_Beam1

This task handles the situations involving breaks on beam 1. There are two copies of this task one for each door and they differ from each other in the mailboxes they retrieve information from. The sensor interrupt handler posts the beam break information to a mailbox on which it pends. If it was the beginning of a possible on count then an entry flag is set. This flag will be tested when next break on beam 1 occurs. If the break occurred and entry flag was set then it is not a valid situation since all beams are to be broken and made again before next break is recognized.

The entry flag is reset by the make_on_beam1 task in case an on count was generated or a make on beam 1 was detected before any other breaks were detected.

D.5.2 Make_on_Beam1

This task complements the task discussed above. It monitors the make on beam1 condition. It is started only by all_beams_broken task after it has detected a break on all four beams. This task too gets the sensor information from the handler and posts to break_on_beam1 task on detecting a make on beam 1. It also posts to on count collection task the newly generated count. Another copy of this task exists for rear door as well.

D.5.3 All_Beams_Broken

This task monitors all four beams for a situation where all four beams are found to be broken. It is activated by entry or exit processing tasks. There are four copies of this task in the system, two each for front and rear doors and one each for entry and exit conditions at each door. The only difference among these four copies is the task that activates it. In case of a situation when an on count is being processed at front door it will be break_on_beam1 task which will activate the copy corresponding to entry at front door. After detecting breaks on all four beams it will post a message to make_on_beam1 task so that it can start monitoring a possible make on that beam.

D.5.4 Break_on_Beam4

This is similar to break_on_beam1 task except that it processes the situations which can potentially generate an off count. The logic is essentially the same, that is, if the first beam to be broken was beam 4 and subsequently all beams were broken and beam 4 was made again then a passenger just got out and an off count generated. It posts to its copy of all_beams_broken to monitor breaking of remaining beams and pends at a mailbox posted to by make_on_beam4 informing it the completion of cycle.

D.5.5 Make_on_Beam4

This task is similar to make_on_beam1 instead that it monitors a make condition on beam 4 after it is activated by all_beams_broken task. It will post to break_on_beam4 task after detecting a make on beam 4. It will also post to collecting task.

D.5.6 Collect_In Task

This task is responsible for collecting the on counts generated by entry processes at a given door. There are two copies of this task which collect on counts for front and rear doors respectively. The counts are passed to them by make_on_beam1 tasks for front and rear doors. This inturn posts these counts to a queue which is read by totalizing task. The collection is allowed only if the corresponding door was detected to be open.

D.5.7 Collect_Out Task

This task is identical to previous one except that it collects off counts being generated at a given door. Two copies of this task exist one each for front and rear doors. This also posts to the same queue as collect_in task which is read by a totalizing task.

D.5.8 Total_In Task

This task accepts on counts from collect_in tasks corresponding to front and rear doors. The only reason this task is provided here is to provide a buffer for collection tasks so that they can concentrate on getting these counts from higher level tasks which have to run in constrained time frames. It continues to totalize the counts as long as vehicle is stationary.

D.5.9 Total_Out Task

It is identical to total_in task except that it totalizes the off counts being generated for both doors.

D.6 Log Generating Tasks

These tasks are responsible for generating necessary information which goes with a particular type of log after being signalled by appropriate task which handles that situation. For example passenger log task is activated by totalizing task whenever there are some counts to be stored after vehicle moves from a stop. These tasks are described below.

D.6.1 Passenger Log Task

The totalizing tasks have the total values of on and off counts since the vehicle stopped. Once the vehicle moves this information is passed to passenger_log task which is responsible for generating appropriate log complete with incremental distance and time values. The log_writer function then can be called upon to store it.

D.6.2 One Minute Idle Log Task

This task is activated by one_minute_idle task whenever it determines that the vehicle has been stationary for a minute. It generates the incremental time and distance information and writes the log.

D.6.3 Two Minute Idle Log Task

This task is activated by two_minute_idle task whenever it determines that the vehicle has been stationary for more than two minutes. It generates the incremental time and distance information and writes the log.

D.6.4 Distance Overflow Log Task

This task is activated by distance overflow task whenever the distance accumulated by it exceeds the value that can be held in a byte. It generates the incremental time information and writes the log.

D.6.5 Hour Overflow Log Task

This task is activated by hour overflow task whenever the ticks accumulated by it exceeds the value equivalent to one hour. It generates the incremental distance information and writes the log.

D.7 Operator Interaction Tasks

The tasks organized under this category are designed to address to various requests that can be made by operator. These could pertain to dumping, displaying or resetting the logs. Most of these tasks are low priority tasks and are given control only when sensor related tasks are dormant.

D.7.1 Get Command Task

This task depends on USART interrupt handler for its activation. The interrupt handler of USART posts the character received from the operator console to this task. This task pends at a mailbox in which character is passed to it. It examines the character to be one of the valid commands and an error is signalled if not. Otherwise this task will activate a lower level task to handle the request by posting appropriate message to the mailboxes at which they pend.

D.7.2 Display Log Task

This task allows the displaying of logs on a terminal in response to a request from the operator. It pends on a mailbox which is posted to by get_command task whenever it detects display command character. This task responds by sending out the logs in 25 byte blocks (5 logs) to the I/O buffer and transmit interrupt handler does the rest.

D.7.3 Dump Log Task

It is similar to the above task except that it keeps sending the logs over the serial channel till all logs have been transmitted. The end of log data is marked by generation of a special set of logs giving the elapsed time, elapsed distance and vehicle identification.

D.7.4 Reset Log Task

This log resets the entire log data memory and initializes it to zeros but only after all logs have been dumped along with total elapsed time and elapsed distance logs.

D.7.5 Diagnostics Task

This task provides an inside look at system parameters of interest to the operator. The data provided includes beam status for both doors, door status and current incremental count values. This task will activate tasks at lower level to transmit the information about these parameters. Note that this information is primarily supplied by interrupt handlers or obtained by reading the I/O ports.

D.7.6 Elapsed Time Log Task

This task generates and writes a log that contains the information regarding total time elapsed since last dump or power on. This information is obtained from the elapsed_time task that maintains it. It can be activated by diagnostics task or dump logs task.

D.7.7 Elapsed Distance Log Task

This task generates and writes a log that contains the information regarding total distance elapsed since last dump or power on. This information is obtained from the elapsed_distance task that maintains it. It can be activated by diagnostics task or dump logs task.

D.7.8 Vehicle ID Log Task

This task generates a log containing information about vehicle identification which is obtained by reading a port. This is activated by diagnostics or dump log tasks.

D.8 Miscellaneous Tasks

Apart from the well defined tasks discussed above some more tasks are required to achieve the desired results. These are as important as any other from system point of view. These are discussed below.

D.8.1 External Input Handling

This task monitors the external input or signpost input as it is normally referred to. This is a high priority task and remains suspended for most of the time, coming into action only after a signpost has been detected. In the present system the signpost signal is derived from the switch used to change the signboard. Upon receiving this signal it will generate a log which will later be used by analysis software to identify beginning of new route.

D.8.2 System Fault Task

This task is a high priority task and monitors the health of the hardware. Most of the calls in the system are time based, that is most of the time requesting tasks have to get the service or resource within a time frame and non availability of such resource may be because of malfunction. It can be invoked by some kind of diagnostic task which will supply the information about fault. Presently only conditions monitored are memory test and time outs generated by tasks.

D.9 Log Structure

A new log structure was devised to incorporate absolute values of time in the logs. This removes the need for incremental time maintenance and time overflow monitoring. It also simplifies the design of off line analysis software. This structure has been given in Table 2 on page 176.

Table 2. Log structure of the new system.

Log type	byte				
	1	2	3	4	5
Power on	1	mm	ss	00	ID
One minute idle	2	mm	ss	dd	00
End of Two minute idle	3	mm	ss	dd	00
Dump elapsed time	4	hh	mm	ss	00
Dump elapsed distance	5	dd	dd	dd	dd
Passenger On	6	mm	ss	dd	On
Passenger Off	7	mm	ss	dd	Off
External input	8	mm	ss	dd	00
Distance overflow	9	mm	ss	00	00
Hour overflow	10	00	00	dd	00
Day change	11	00	00	dd	00
Distance overrun	12	mm	ss	dd	00
System fault	13	xx	xx	xx	xx

mm	Absolute value of minutes
ss	Absolute value of seconds
hh	Absolute value of hours
dd	Incremental distance since last log
xx	Function/device identification

Appendix E. Sensors

Sensors play a very important role in any real time system since they provide all the information about the external events to the processor and carry its outputs to outside world. It will not be wrong to say that sensors are the ears and eyes of a real time computer system. The sensors or transducers are devices that convert one form of energy into another form. These transducers may draw upon many of the properties of materials or characteristics of physical phenomena to achieve this transformation. For example the quartz crystal which converts the pressure into electric charge due to piezo-electric effect.

The examples of sensors abound in day to day life as well as in industry. Virtually any kind of real world interfacing would mandate some kind of transducer. However the discussion in this section is confined to just one type of transducers which can convert light into electricity. The principle of operation of these sensors is fairly straight forward. Whenever light of sufficient intensity is incident on these they produce an electric current. In order to make successful use of such devices it is desirable to control both the source and the intensity of light. It is done by designing an optical source and a matching detector. The source normally provides a narrow beam of infrared light which can hit through the aperture of the detector to trigger it.

These sensors are used in pairs of transmitters and receiver. The source when powered up generates the infrared beam which triggers the detector when properly aligned. Now if the beam is broken by an object in its path the detector will not receive the beam and will consequently switch off. Thus presence or absence of the current at the detector gives a sure indication if there is an object between the transmitter and receiver. These kinds of sensors are used frequently for counting and position detection applications.

The system designed here also uses similar sensors that is transmitter and receiver pairs for counting passengers boarding and alighting the vehicle. Each doorwell is equipped with four such pairs providing four beams. The problem with this kind of arrangement is that ambient light can cause false detection and detection sensitivity varies significantly with the separation between the transmitter and receiver. The sensors used in this application are the same as used with Red Pine System. A detailed study and analysis of those sensors was undertaken in order to understand the sensor behavior and their impact on overall system performance.

E.1 Fiber Optic Sensors

The recent advances in Fiber Optic Sensing Systems (FOSS) technology has made it feasible to consider fiber optic sensors for use in such applications. The type of sensors most suitable for this kind of applications are distributed fiber optic sensor and microbend sensors [18]. The fibers may be embedded on the stairs of door well in a well defined pattern to detect the pressure exerted on them by moving passengers. It may also be possible to somehow combine these optical signals to produce directional information as well. This is a growing field and there are plenty of possibilities.

Appendix F. Input Output Interfacing

In order to provide the actual field information to the processor it is necessary that all input points are interfaced with the processor in some fashion. It is recommended that all signals coming from the field pass through a signal conditioning stage before getting inside the processor environment. The main purpose of signal conditioning is to filter out the noise and perform level translation if needed.

In the case of present application there are eight pairs of sensors, one signpost input, one distance transducer input and two door status signals. The signals coming from the door are of on/off type and present a level. If these signals are coming from a relay contact then there has to be a time delay circuit at the front to debounce the contact and filter switching noise. If the relay contacts do not give 5 volt signal then a level translator may be required. Often optoisolators are used for level translation which also provide input and output isolation protecting the processor from damage by spikes on input lines. The signpost signal which is a pulse signal is interfaced in a similar fashion.

The block diagram shown in Figure 43 on page 180 shows the interfacing technique used to connect field signals to the processor.

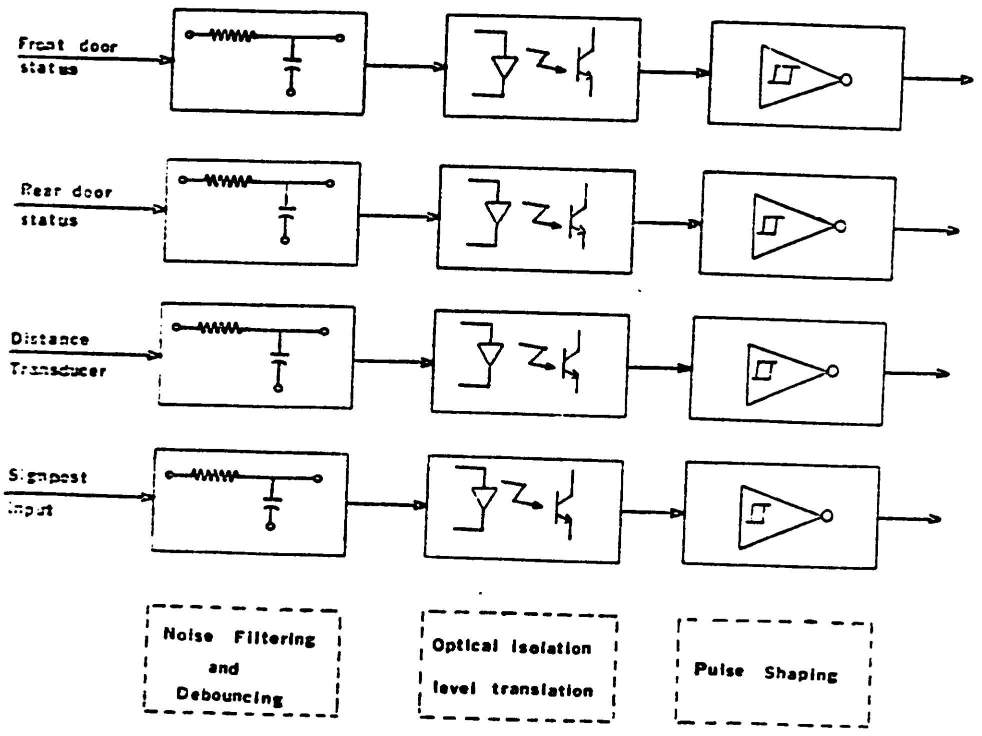


Figure 43. I/O Interfacing and Signal Conditioning

The sensor used in this type of system produces 4000 pulses for every eight feet. These pulses are normally distorted and noisy hence a filtering stage is required. A filter followed by an optoisolator provides an isolated and comparatively noise free pulse but it loses its amplitude and the driving capacity. The output of optoisolator can be used to drive a schmitt trigger circuit which will regenerate a noise free pulse with the amplitude of 5 volts which can then be safely fed to microcomputer input circuits.

A detailed discussion about the sensors have been provided in Appendix-E. The number of lines coming into processor circuits are dependent on the detection scheme used. The detector circuit can provide one of the following possibilities.

1. One line output which is pulsed whenever a sensor pair is affected. This means that whenever one or more beams are made or broken an interrupt is generated.

2. Eight output lines corresponding to each pair. The lines corresponding to one or more pairs are pulsed whenever their respective beams are broken or made.

3. Sixteen output lines, two lines per pair. One line corresponds to beam make interrupt and another to beam break interrupt.

The cpu has to read the status of beams through a port to determine as to which sensor pair or pairs were responsible for interrupt in the first case. In second case the cpu has to decide as to what type (make or break) of interrupt occurred since it already knows which pair caused it. In third case it knows exactly what happened by the interrupt. The first approach is highly software intensive and though cheap may result in poor response time. The third approach is highly hardware intensive and hence more expensive as well as less reliable. The second approach offers a good mix of hardware and software. An 82C59A PIC may be used to gate all this interrupts so that cpu can branch to corresponding service routines.

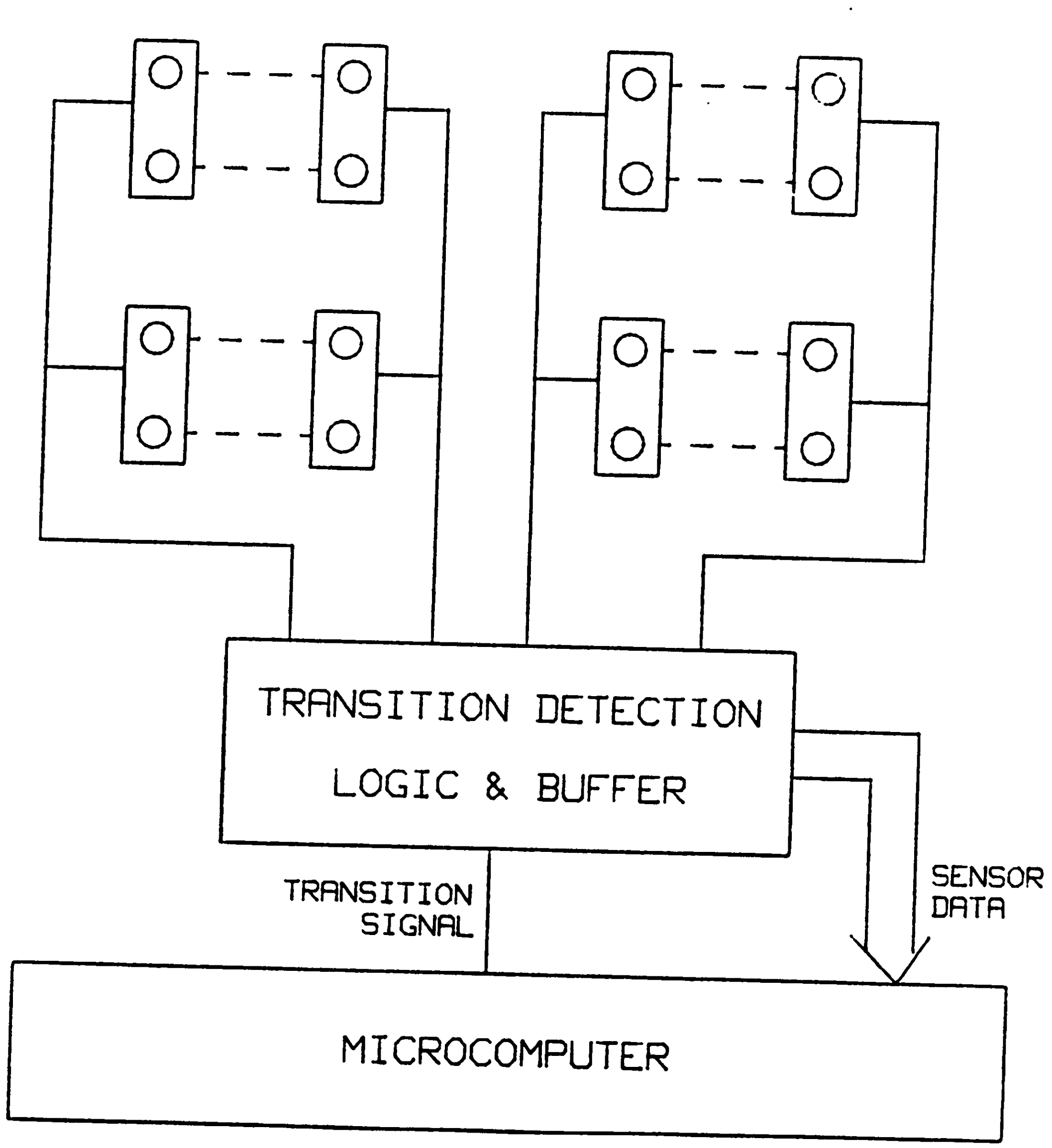


Figure 44. Sensor Interfacing and Signal Conditioning

The block diagram shown in Figure 44 on page 182 shows the interfacing technique used in the present case. In our case however the first approach was used in order to save on the extra board required which would increase the power and space requirements. Moreover only first approach lends itself easily to simulation which was a top priority since the system is to be tested for logical correctness first.

These circuits were actually built and tested using toggle and momentary switches. They were also incorporated on the simulation board which was interfaced with the microcomputer to test part of the application program.
Figure 45 on page 184 shows the block diagram of the simulation board set up for testing the software. Each sensor pair was simulated by means of a toggle switch and interrupts were generated via a momentary switch. Toggle switches were also used to simulate door control and signpost signals.

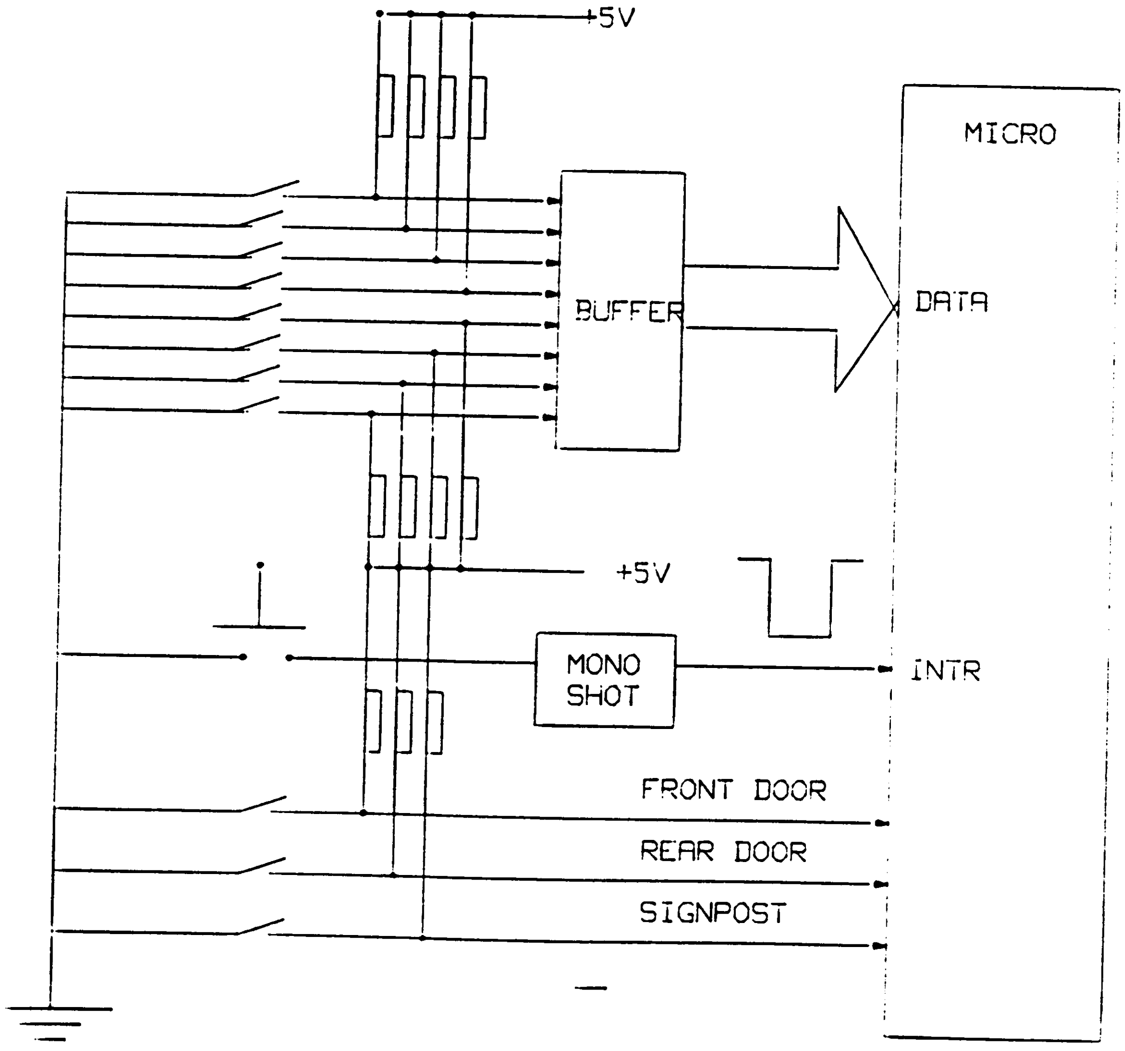


Figure 45. Block Diagram of Simulation Board

Appendix G. Board Support Package Listings

This section includes the source code for the board support package developed for MSI-C988 microcomputer board as well as the source code for the test programs developed to test this software. Notice that the test program included here actually implements some of the functions incorporated in proposed multitasking vehicular data acquisition system design. Please contact Dr. Charles E. Nunnally, Professor in Electrical Engineering Department at Virginia Tech, for the source code of individual tasks and the entire application software developed for this application.

The first module contains the initialization procedures and interrupt service routines for all the devices on MSI-C988 board. The second module is provided as an application example. It consists of tasks to exercise most of the board support software. Users can develop their own application or test software along the same lines. The listings provide the details of its functioning.

```
             ;*********************************************
             ;*                                           *
             ;*          Board Support Package            *
             ;*                   for                     *
             ;*      MSI-C988 Microcomputer Board         *
             ;*                                           *
             ;*      Author : Pradyumna K Misra           *
             ;*      Date   : June 1987                   *
             ;*                                           *
             ;*********************************************

                     NAME    C988_BSP

             ; All variables declared externally to this
             ; module are contained in EXTNL.INC file.

           C         INCLUDE EXTNL.INC
           C ;*********************************************
           C ;                                            *
           C ;       Externally Declared Variables        *
           C ;                                            *
           C ;*********************************************
           C
           C ; Configuration Table
           C
           C         EXTRN   TBL:BYTE
           C
           C ; Variables to hold sensor beam data
           C
           C         EXTRN   B_STAT:WORD
           C         EXTRN   LAST_WD:WORD
           C
           C ; Procedure MAIN to create remaining tasks
           C
           C         EXTRN   MAIN:FAR
           C
           C ; Mailbox for incremental distance
           C
           C         EXTRN   DIST:DWORD
           C
           C ; Mailboxes to be used by odometer interrupts
           C
           C         EXTRN   MBOX_OD1:DWORD
           C         EXTRN   MBOX_OD2:DWORD
           C         EXTRN   MBOX_OD3:DWORD
           C         EXTRN   MBOX_OD4:DWORD
           C         EXTRN   MBOX_OD5:DWORD
           C
           C ; Mailbox used by external input interrupts
           C
           C         EXTRN   SIGN_BOX:DWORD
```

```
                              C
                              C  ; Mailboxes used by sensor interrupts
                              C
                              C  ; Mailboxes to report breaking of beams
                              C
                              C         EXTRN   MBOX_E1:DWORD
                              C         EXTRN   MBOX_E2:DWORD
                              C         EXTRN   MBOX_E3:DWORD
                              C         EXTRN   MBOX_E4:DWORD
                              C         EXTRN   MBOX_E5:DWORD
                              C         EXTRN   MBOX_E6:DWORD
                              C         EXTRN   MBOX_E7:DWORD
                              C         EXTRN   MBOX_E8:DWORD
                              C
                              C  ; Mailboxes to report making of beams
                              C
                              C         EXTRN   MBOX_E9:DWORD
                              C         EXTRN   MBOX_E10:DWORD
                              C         EXTRN   MBOX_E11:DWORD
                              C         EXTRN   MBOX_E12:DWORD
                              C         EXTRN   MBOX_E13:DWORD
                              C         EXTRN   MBOX_E14:DWORD
                              C         EXTRN   MBOX_E15:DWORD
                              C         EXTRN   MBOX_E16:DWORD
                              C


                                 ; All VRTX system calls have been equated
                                 ; in the include file VRTX.INC.

                              C         INCLUDE VRTX.INC
                              C
                              C  ; VRTX header file (SCCS 1.5)
                              C
 = 0020                       C  VRTX       EQU   20H ; VRTX interrupt number
                              C
                              C  ; VRTX task-level function codes
                              C
 = 0000                       C  SC_TCREATE   EQU 0000H ; create a task
 = 0001                       C  SC_TDELETE   EQU 0001H ; task delete
 = 0002                       C  SC_TSUSPEND  EQU 0002H ; task suspend
 = 0003                       C  SC_TRESUME   EQU 0003H ; task resume
 = 0004                       C  SC_TPRIORITY EQU 0004H ; task priority change
 = 0005                       C  SC_TINQUIRY  EQU 0005H ; task inquiry
 = 0006                       C  SC_GBLOCK    EQU 0006H ; get memory block
 = 0007                       C  SC_RBLOCK    EQU 0007H ; release memory block
 = 0008                       C  SC_POST      EQU 0008H ; post a message
 = 0009                       C  SC_PEND      EQU 0009H ; pend for a message
 = 000A                       C  SC_GTIME     EQU 000AH ; get time
 = 000B                       C  SC_STIME     EQU 000BH ; set time
 = 000C                       C  SC_TDELAY    EQU 000CH ; task delay
 = 000D                       C  SC_GETC      EQU 000DH ; get character
 = 000E                       C  SC_PUTC      EQU 000EH ; put character
 = 000F                       C  SC_WAITC     EQU 000FH ; wait for special chara
```

```
                                  cter
= 0015                          C SC_TSLICE     EQU 0015H ; enable round-robin sch
                                  eduling
= 0020                          C SC_LOCK       EQU 0020H ; disable task reschedul
                                  ing
= 0021                          C SC_UNLOCK     EQU 0021H ; enable task rescheduli
                                  ng
= 0022                          C SC_PCREATE    EQU 0022H ; create memory partitio
                                  n
= 0023                          C SC_PEXTEND    EQU 0023H ; extend memory partitio
                                  n
= 0025                          C SC_ACCEPT     EQU 0025H ; accept a message
= 0026                          C SC_QPOST      EQU 0026H ; post message to queue
= 0027                          C SC_QPEND      EQU 0027H ; pend for message from
                                  queue
= 0028                          C SC_QACCEPT    EQU 0028H ; accept message from qu
                                  eue
= 0029                          C SC_QCREATE    EQU 0029H ; create message queue
= 002A                          C SC_QINQUIRY   EQU 002AH ; queue inquiry
                                C
                                C ; VRTX interrupt-level function codes
                                C
= 0011                          C UI_EXIT     EQU 0011H ; exit from interrupt hand
                                  ler
= 0012                          C UI_TIMER    EQU 0012H ; post time increment from
                                    interrupt
= 0013                          C UI_RXCHR    EQU 0013H ; post received char from
                                  interrupt
= 0014                          C UI_TXRDY    EQU 0014H ; post transmit ready from
                                    interrupt
= 0016                          C UI_ENTER    EQU 0016H ; enter interrupt handler
                                C
                                C ; VRTX initialization function codes
                                C
= 0030                          C VRTX_INIT   EQU 0030H ; initialize VRTX
= 0031                          C VRTX_GO     EQU 0031H ; start multitasking
                                C
                                C ; System-wide error codes
                                C
= 0000                          C RET_OK      EQU 0000H ; successful return
                                C
                                C ; VRTX error codes
                                C
= 0001                          C ER_TID      EQU 0001H ; task ID error
= 0002                          C ER_TCB      EQU 0002H ; no TCBs available
= 0003                          C ER_MEM      EQU 0003H ; no memory available
= 0004                          C ER_NMB      EQU 0004H ; not a memory block
= 0005                          C ER_MIU      EQU 0005H ; mailbox in use
= 0006                          C ER_ZMW      EQU 0006H ; zero message
= 0007                          C ER_BUF      EQU 0007H ; buffer full
= 0008                          C ER_WTC      EQU 0008H ; WAITC in progress
= 0009                          C ER_ISC      EQU 0009H ; invalid system call
= 000A                          C ER_TMO      EQU 000AH ; time-out
= 000B                          C ER_NMP      EQU 000BH ; no message present
```

```
 = 000C                          C  ER_QID     EQU 000CH ; queue ID error
 = 000D                          C  ER_QFL     EQU 000DH ; queue full
 = 000E                          C  ER_PID     EQU 000EH ; partition ID error
 = 000F                          C  ER_INI     EQU 000FH ; fatal initialization erro
                                    r
 = 0010                          C  ER_NCP     EQU 0010H ; no character present
 = 0020                          C  ER_CVT     EQU 0020H ; component vector table no
                                    t present
 = 0021                          C  ER_COM     EQU 0021H ; undefined component
 = 0022                          C  ER_OPC     EQU 0022H ; undefined opcode for comp
                                    onent
```

```
                                page
 0000                           BSP_CODE SEGMENT PARA PUBLIC 'BSP_CODE'
                                         ASSUME CS:BSP_CODE,DS:BSP_CODE

                                ;****************************************
                                ;                                      *
                                ;       PRE INITIALIZATION CODE        *
                                ;                                      *
                                ;****************************************

                                ;!!!!!!!!!!!!!  IMPORTANT  !!!!!!!!!!!!!

                                ;****************************************
                                ;*                                     *
                                ;*  VRTX assumes that pointer to       *
                                ;*  CFTBL resides at 0200 (int 128)    *
                                ;*  But in our PROM it has been        *
                                ;*  changed to 0184H for use with PC   *
                                ;*                                     *
                                ;****************************************

                                ; Locations for offset and segment
                                ; of pointers to configuration table.

 = 0184                         CFTBL_PTR_OFF    EQU WORD PTR 0184H
 = 0186                         CFTBL_PTR_SEG    EQU WORD PTR 0186H

                                ; Setup VRTX entry point offset and segment.
                                ; Physical address of VRTX PROM is 0A000 Hex

 = 0000                         VIP              EQU      00000H
 = 0A00                         VCS              EQU      00A00H

                                ; Interrupt vector corresponding to INT 32 of
                                ; 8086/88 Interrupt Vector Table is chosen as
                                ; interrupt to make VRTX system calls

 = 0020                         VRTX             EQU WORD PTR 020H
 = 0080                         INT32_VCT        EQU WORD PTR 080H
```

```
                              page
                              ; Initialization process starts here

 0000                         INIT          PROC    FAR

                              ; Set DS to point to IVT
                              ; Load in pointer to configuration table
                              ; Jump to VRIX initialize entry point

 0000  B8 0000                ENTRY:  MOV     AX, 0
 0003  8E D8                          MOV     DS, AX
 0005  C7 06 0186 ---- E              MOV     DS:CFTBL_PTR_SEG, SEG TBL
 000B  C7 06 0184 0000 E              MOV     DS:CFTBL_PTR_OFF, OFFSET TBL

                              ; Load interrupt vector 32 (20H)
                              ; for VRIX system call

 0011  B8 0000                        MOV     AX, VIP
 0014  A3 0080                        MOV     DS:INT32_VCT, AX
 0017  B8 0A00                        MOV     AX, VCS
 001A  A3 0082                        MOV     DS:INT32_VCT+2, AX

                              ; Issue VRIX_INIT call to initialize VRIX

 001D  B8 0030                        MOV     AX, VRIX_INIT
 0020  CD 20                          INT     VRIX
 0022  3D 0000                        CMP     AX, RET_OK
 0025  74 02                          JZ      P_INIT

                              ; Non zero return code => error

 0027  EB D7                  ERROR:  JMP     ENTRY

                              ;       POST INITIALIZATION

 0029  E8 0051 R              P_INIT: CALL    USART_INIT
 002C  E8 013D R                      CALL    PIC_INIT
 002F  E8 014E R                      CALL    PIT_INIT
 0032  E8 0228 R                      CALL    SGN_INIT
 0035  E8 00C7 R                      CALL    USART2
 0038  E8 0258 R                      CALL    RTC
```

```
                                   page
                                   ; Create first task (PARENT PROCESS)
                                   ; Must be highest priority process

                                   ; id = 0  pri = 0
 003B  BB 0000 E                           MOV     BX, OFFSET MAIN
 003E  B8 ---- E                           MOV     AX, SEG MAIN
 0041  8E C0                               MOV     ES, AX
 0043  B1 00                               MOV     CL, 0           ; id = 0
 0045  B5 00                               MOV     CH, 0           ; pri = 0
 0047  B8 0000                             MOV     AX, SC_TCREATE
 004A  CD 20                               INT     VRTX

                                   ; Issue VRTX GO system call
                                   ; to start multitasking

 004C  B8 0031                             MOV     AX, VRTX_GO
 004F  CD 20                               INT     VRTX

                                   ; No return from this call

                                   INIT            ENDP
```

```
                                page
                                ;****************************************
                                ;                                       *
                                ;  8252 USART Initialization Procedure  *
                                ;                                       *
                                ;****************************************

= 0001                                  UCR             EQU     01H
= 0003                                  BRSR            EQU     03H
= 0002                                  MCR             EQU     02H
= 0000                                  USART_DATA      EQU     00H
= 0024                                  INT09_VCT       EQU     WORD PTR 024H
= 0028                                  INT10_VCT       EQU     WORD PTR 028H

                                ; Monitor initialization is left intact

                                ; Control word for Usart Control Register 03CH
                                ; => 8 bits/character, no parity, 1 stop bit

                                ; Control word for Baud Rate Register 0EH
                                ; => Baud rate 4800

                                ; Control word for Modem Control Register 023H
                                ; => Receiver Enabled, DTR = DSR = 0
                                ;    USART and MODEM interrupts disabled

 0051                           USART_INIT      PROC    NEAR

 0051  BA 0001                          MOV     DX, UCR
 0054  B0 3C                            MOV     AL, 03CH
 0056  EE                               OUT     DX, AL
 0057  BA 0003                          MOV     DX, BRSR
 005A  B0 0E                            MOV     AL, 0EH
 005C  EE                               OUT     DX, AL
 005D  BA 0002                          MOV     DX, MCR
 0060  B0 23                            MOV     AL, 023H
 0062  EE                               OUT     DX, AL
 0063  BA 0000                          MOV     DX, 0
 0066  EC                               IN      AL, DX          ; Dummy read

                                ; SET INTERRUPT VECTORS
                                ; For receive and transmit service routines

 0067  B8 0080 R                        MOV     AX, OFFSET CIO_RX
 006A  A3 0024                          MOV     DS:INT09_VCT, AX
 006D  B8 ---- R                        MOV     AX, SEG CIO_RX
 0070  A3 0026                          MOV     DS:INT09_VCT+2, AX
 0073  B8 009E R                        MOV     AX, OFFSET CIO_TX
 0076  A3 0028                          MOV     DS:INT10_VCT, AX
 0079  B8 ---- R                        MOV     AX, SEG CIO_TX
 007C  A3 002A                          MOV     DS:INT10_VCT+2, AX

 007F  C3                               RET
```

```
                          USART_INIT      ENDP


                          ;*******************************************
                          ;                                          *
                          ; USART Character Receive Interrupt Handler *
                          ;                                          *
                          ;*******************************************

0080                      CIO_RX          PROC    FAR

                          ; enter system mode

0080  50                         PUSH    AX               ; Save AX
0081  B8 0016                    MOV     AX, UI_ENTER     ; ui_enter
0084  CD 20                      INT     VRTX             ; call VRTX

                          ; save registers used
                          ; get character from USART and pass to VRTX
                          ; restore registers

0086  51                         PUSH    CX               ; save cx
0087  E5 00                      IN      AX, 0            ; input char to ax
0089  24 7F                      AND     AL, 07FH
008B  8A E8                      MOV     CH, AL           ; VRTX expects character
                                                          ; in register CH
008D  B8 0013                    MOV     AX, UI_RXCHR     ; ui_rxchr
0090  CD 20                      INT     VRTX             ; call VRTX
0092  59                         POP     CX               ; restore cx

                          ; send EOI to 82C59A PIC

0093  B0 20                      MOV     AL,020H
0095  BA 0040                    MOV     DX,040H
0098  EE                         OUT     DX,AL

                          ; exit system mode

0099  B8 0011                    MOV     AX, UI_EXIT      ; ui_exit
009C  CD 20                      INT     VRTX             ; call VRTX
                                                          ; VRTX restores AX

                          CIO_RX          ENDP

                          ;******************************************
                          ;                                         *
                          ;USART Character Transmit Interrupt Handler*
                          ;                                         *
                          ;******************************************
```

```
 009E                           CIO_TX          PROC    FAR
                                ; enter system mode
 009E  50                               PUSH    AX              ; save registers
 009F  51                               PUSH    CX

 00A0  B8 0016                          MOV     AX, UI_ENTER
 00A3  CD 20                            INT     VRTX

                                ; request next character for printing
 00A5  B8 0014                          MOV     AX, UI_TXRDY
 00A8  CD 20                            INT     VRTX

                                ; Test If character returned
                                ; Exit if no character (AX not equal 0)
 00AA  3D 0000                          CMP     AX, RET_OK
 00AD  75 05                            JNZ     EXT

                                ; Call driver to transmit character
 00AF  9A 00C0 ---- R                   CALL    FAR PTR TXRDY

                                ; exit system mode
                                ; SEND EOI TO PIC

 00B4  B0 20                    EXT:    MOV     AL, 020H
 00B6  BA 0040                          MOV     DX, 040H
 00B9  EE                               OUT     DX, AL

 00BA  59                               POP     CX              ; restore CX
 00BB  B8 0011                          MOV     AX, UI_EXIT
 00BE  CD 20                            INT     VRTX            ; VRTX restores

                                CIO_TX          ENDP

                                ;******************************************
                                ;*                                        *
                                ;*              TXRDY Driver              *
                                ;*                                        *
                                ;******************************************

                                                PUBLIC  TXRDY

 00C0                           TXRDY           PROC    FAR

 00C0  8A C5                            MOV     AL, CH          ; Get character
 00C2  BA 0000                          MOV     DX, 0
 00C5  EE                               OUT     DX, AL          ; output to usart
```

```
                                                       ; data port
 00C6  CB                              RET
                                 ; Return to caller, ISR or VRTX
                                 TXRDY           ENDP

                                 ;****************************************
                                 ;                                       *
                                 ; 8252 USART#2 Initialization Procedure *
                                 ;                                       *
                                 ;****************************************

 = 0021                                UCR2            EQU     21H
 = 0023                                BRSR2           EQU     23H
 = 0022                                MCR2            EQU     22H
 = 0020                                UDATA2          EQU     20H
 = 0038                                INT14_VCT       EQU     WORD PTR 038H
 = 003C                                INT15_VCT       EQU     WORD PTR 03CH

                                 ; USART#2 is initialized exactly in the same
                                 ; way with the same parameters as USART#1

 00C7                            USART2          PROC    NEAR

 00C7  BA 0021                         MOV     DX, UCR2
 00CA  B0 3C                           MOV     AL, 03CH
 00CC  EE                              OUT     DX, AL
 00CD  BA 0023                         MOV     DX, BRSR2
 00D0  B0 0E                           MOV     AL, 0EH
 00D2  EE                              OUT     DX, AL
 00D3  BA 0022                         MOV     DX, MCR2
 00D6  B0 23                           MOV     AL, 023H
 00D8  EE                              OUT     DX, AL
 00D9  BA 0000                         MOV     DX, 0           ; Dummy Read
 00DC  EC                              IN      AL, DX

                                 ; SET INTERRUPT VECTORS

 00DD  B8 00F6 R                       MOV     AX, OFFSET CIN
 00E0  A3 0038                         MOV     DS:INT14_VCT, AX
 00E3  B8 ---- R                       MOV     AX, SEG CIN
 00E6  A3 003A                         MOV     DS:INT14_VCT+2, AX
 00E9  B8 0114 R                       MOV     AX, OFFSET COUT
 00EC  A3 003C                         MOV     DS:INT15_VCT, AX
 00EF  B8 ---- R                       MOV     AX, SEG COUT
 00F2  A3 003E                         MOV     DS:INT15_VCT+2, AX

 00F5  C3                              RET

                                 USART2          ENDP
```

```
                              ;********************************************
                              ;                                           *
                              ; USART2 Character Receive Interrupt Handler *
                              ;                                           *
                              ;********************************************

 00F6                         CIN             PROC    FAR

                              ; enter system mode

 00F6  50                             PUSH    AX
 00F7  B8 0016                        MOV     AX, UI_ENTER
 00FA  CD 20                          INT     VRTX

                              ; save registers used
                              ; get character from USART and pass to VRTX
                              ; restore registers

 00FC  51                             PUSH    CX
 00FD  E5 00                          IN      AX, 0
 00FF  24 7F                          AND     AL, 07FH
 0101  8A E8                          MOV     CH, AL
 0103  B8 0013                        MOV     AX, UI_RXCHR
 0106  CD 20                          INT     VRTX
 0108  59                             POP     CX

                              ; send EOI to 82C59A PIC

 0109  B0 20                          MOV     AL,020H
 010B  BA 0040                        MOV     DX,040H
 010E  EE                             OUT     DX,AL

                              ; exit system mode

 010F  B8 0011                        MOV     AX, UI_EXIT
 0112  CD 20                          INT     VRTX

                              CIN             ENDP

                              ;******************************************
                              ;                                         *
                              ;     USART2 Character Transmit ISR       *
                              ;                                         *
                              ;******************************************

 0114                         COUT            PROC    FAR

                              ; enter system mode

 0114  50                             PUSH    AX
 0115  51                             PUSH    CX
```

```
0116  B8 0016                           MOV     AX, UI_ENTER
0119  CD 20                             INT     VRTX

                                ; request next character for printing

011B  B8 0014                           MOV     AX, UI_TXRDY
011E  CD 20                             INT     VRTX

                                ; Test uf character returned
                                ; Exit if no character (AX not equal 0)

0120  3D 0000                           CMP     AX, RET_OK
0123  75 05                             JNZ     EXT2

                                ; Call driver to transmit character

0125  9A 0136 ---- R                    CALL    FAR PTR TX_RDY

                                ; exit system mode

                                ; SEND EOI TO PIC

012A  B0 20                     EXT2:   MOV     AL, 020H
012C  BA 0040                           MOV     DX, 040H
012F  EE                                OUT     DX, AL

0130  59                                POP     CX
0131  B8 0011                           MOV     AX, UI_EXIT
0134  CD 20                             INT     VRTX

                                COUT            ENDP

                                ;*******************************************
                                ;*                                         *
                                ;*      USART2  TX_RDY Driver              *
                                ;*                                         *
                                ;*******************************************

                                                PUBLIC  TX_RDY

0136                            TX_RDY          PROC    FAR

0136  8A C5                             MOV     AL, CH          ; Get character
0138  BA 0020                           MOV     DX, 020H
013B  EE                                OUT     DX, AL          ; output to usart
                                                                ; data port
013C  CB                                RET
                                ; Return to caller, ISR or VRTX

                                TX_RDY          ENDP
```

```
                               ;****************************************
                               ;                                        *
                               ;  82C59 PIC Initialization Procedure    *
                               ;                                        *
                               ;****************************************

                               ; Same initialization as in monitor except both

                               ; receive and transmit interrupts are enabled
                               ; right here instead by callin routines.

                               ; Doing 8259 initialization to enable both
                               ; tx and rx interrupts

= 0040                                ICW1      EQU     040H
= 0041                                ICW2      EQU     041H
= 0041                                ICW3      EQU     041H
= 0041                                ICW4      EQU     041H
= 0041                                OCW1      EQU     041H
= 0040                                OCW2      EQU     040H
= 0040                                OCW3      EQU     040H

 013D                          PIC_INIT                PROC            NEAR

                               ; PIC Initialization parameters
                               ; Edge Triggered, INT=4, Single 8259
                               ; ICW4 to follow

 013D  B0 17                          MOV       AL, 017H       ; 0001 0111
 013F  E6 40                          OUT       ICW1,AL

                               ; INT type 8 (starts at 020H)

 0141  B0 08                          MOV       AL, 08H        ; INT TYPE 8
 0143  E6 41                          OUT       ICW2, AL

                               ; ICW4 => SFNM=0, Buffered mode/Master
                               ; Normal End-of-Interrupt (EOI), 8086/88 mode

 0145  B0 0D                          MOV       AL, 0DH
 0147  E6 41                          OUT       ICW2, AL

                               ; Interrupt Mask 11100000 Enables IR0 thru IR4

                               ; 5 interrupts corresponding to
                               ; USART#1, Timer, Odometer and External Input
                               ; USART#2 and Sensor interrupts disabled for
                               ; the TEST program which tests this
                               ; Board Support Package and does not use them.

 0149  B0 E0                          MOV       AL, 0E0H
 014B  E6 41                          OUT       ICW4, AL
```

```
014D  C3                             RET
                              ; Return to init routine
                              PIC_INIT         ENDP

                              ;*********************************************
                              ;*                                           *
                              ;*       TIMER INITIALIZATION -82C54         *
                              ;*                                           *
                              ;*********************************************
= 0063                               PIT_WCR EQU     063H    ; CONTROL REG
= 0060                               CTR_0   EQU     060H    ; COUNTER 0
= 0061                               CTR_1   EQU     061H    ; COUNTER 1
= 0062                               CTR_2   EQU     062H    ; COUNTER 3

= 0020                               INT8_VCT        EQU     WORD PTR 020H
= 002C                               INT11_VCT       EQU     WORD PTR 02CH

014E                          PIT_INIT         PROC    NEAR

                              ; Counter 0 is programmed for 10 msec interval

                              ; timer to be used as Clock for VRTX
                              ; Control word 034H
                              ; => counter0, mode2, binary counting, lsb/msb

                              ; On C988 board connect following jumpers
                              ; pin 20 (Clk0) to pin 22 (Pclk)
                              ; pin 19 (Out0) to pin a  (8259 IR0)

                              ; Counters 1 and 2 are programmed for
                              ; divide by N in Mode 2 to count
                              ; Odometer pulses fed from outside.

014E  BA 0063                        MOV     DX, PIT_WCR
0151  B0 34                          MOV     AL, 034H
0153  EE                             OUT     DX, AL

                              ; 10 ms correspond to 100 Hz. PCLK=2.5 MHz
                              ; 100Hz = 2.5MHz/25000 : 25000D = 061A8H

0154  BA 0060                        MOV     DX, CTR_0
0157  B0 A8                          MOV     AL, 0A8H
0159  EE                             OUT     DX, AL
015A  B0 61                          MOV     AL, 061H
015C  EE                             OUT     DX, AL

                              ;set interrupt vector (int 8) for clock

015D  B8 0194 R                      MOV     AX, OFFSET CLOCK
```

```
0160  A3 0020                        MOV     DS:INT8_VCT, AX
0163  B8 ---- R                      MOV     AX, SEG CLOCK
0166  A3 0022                        MOV     DS:INT8_VCT+2, AX

                               ; Program remaining two counters
                               ; Control word 074H
                               ; => counter1, mode2, binary, lsb/msb
                               ; divide by 300D = 012CH

0169  BA 0063                        MOV     DX, PIT_WCR
016C  B0 74                          MOV     AL, 074H
016E  EE                             OUT     DX, AL
016F  BA 0061                        MOV     DX, CTR_1
0172  B0 2C                          MOV     AL, 02CH
0174  EE                             OUT     DX, AL
0175  B0 01                          MOV     AL, 01H
0177  EE                             OUT     DX, AL

                               ; Control word B4H
                               ; => counter2, mode2, binary, lsb/msb
                               ; divide by 1 (temporarily)

0178  BA 0063                        MOV     DX, PIT_WCR
017B  B0 B4                          MOV     AL, 0B4H
017D  EE                             OUT     DX, AL
017E  BA 0062                        MOV     DX, CTR_2
0181  B0 01                          MOV     AL, 1
0183  EE                             OUT     DX, AL
0184  B0 00                          MOV     AL, 0
0186  EE                             OUT     DX, AL

                               ; Set interrupt vector (INT 11) for Odometer

0187  B8 01AA R                      MOV     AX, OFFSET ODOMETER
018A  A3 002C                        MOV     DS:INT11_VCT, AX
018D  B8 ---- R                      MOV     AX, SEG ODOMETER
0190  A3 002E                        MOV     DS:INT11_VCT+2, AX

0193  C3                             RET

                               PIT_INIT        ENDP

                               ;****************************************
                               ;*                                      *
                               ;*      8254 COUNTER INT HANDLER        *
                               ;*                                      *
                               ;****************************************

0194                           CLOCK   PROC    FAR

                               ; enter system mode

0194  50                             PUSH    AX
```

```
0195  B8 0016                          MOV     AX, UI_ENTER
0198  CD 20                            INT     VRTX

                                 ; announce another tick to VRTX

019A  B8 0012                          MOV     AX, 012H
019D  CD 20                            INT     VRTX

                                 ;send EOI to 8259

019F  BA 0040                          MOV     DX, 040H
01A2  B0 20                            MOV     AL, 020H
01A4  EE                               OUT     DX, AL

                                 ; exit system mode

01A5  B8 0011                          MOV     AX, UI_EXIT      ; UI_EXIT
01A8  CD 20                            INT     VRTX

                                 CLOCK   ENDP

                                 ;****************************************
                                 ;*                                      *
                                 ;*          ISR : ODOMETER              *
                                 ;*                                      *
                                 ;****************************************

                                 ; The odometer pulses are fed to 82C54 PIT
                                 ; running in binary divide by N counter mode.
                                 ; The OUT pin of PIT is connected to INT 5
                                 ; line of 82C59A PIC.

01AA                             ODOMETER      PROC           FAR

01AA  50                               PUSH    AX
01AB  B8 0016                          MOV     AX, UI_ENTER
01AE  CD 20                            INT     VRTX

                                 ; Post Non zero messages to all five mailboxes
                                 ; to intimate pending tasks of odometer pulse

01B0  BB 0000 E                        MOV     BX, OFFSET MBOX_OD1
01B3  B8 ---- E                        MOV     AX, SEG MBOX_OD1
01B6  8E C0                            MOV     ES, AX
01B8  B9 0001                          MOV     CX, 1
01BB  2B D2                            SUB     DX, DX
01BD  B8 0008                          MOV     AX, SC_POST
01C0  CD 20                            INT     VRTX

01C2  BB 0000 E                        MOV     BX, OFFSET MBOX_OD2
01C5  B8 ---- E                        MOV     AX, SEG MBOX_OD2
01C8  8E C0                            MOV     ES, AX
01CA  B9 0001                          MOV     CX, 1
01CD  2B D2                            SUB     DX, DX
```

```
01CF  B8 0008                        MOV     AX, SC_POST
01D2  CD 20                          INT     VRTX

01D4  BB 0000 E                      MOV     BX, OFFSET MBOX_OD3
01D7  B8 ---- E                      MOV     AX, SEG MBOX_OD3
01DA  8E C0                          MOV     ES, AX
01DC  B9 0001                        MOV     CX, 1
01DF  2B D2                          SUB     DX, DX
01E1  B8 0008                        MOV     AX, SC_POST
01E4  CD 20                          INT     VRTX

01E6  BB 0000 E                      MOV     BX, OFFSET MBOX_OD4
01E9  B8 ---- E                      MOV     AX, SEG MBOX_OD4
01EC  8E C0                          MOV     ES, AX
01EE  B9 0001                        MOV     CX, 1
01F1  2B D2                          SUB     DX, DX
01F3  B8 0008                        MOV     AX, SC_POST
01F6  CD 20                          INT     VRTX

01F8  BB 0000 E                      MOV     BX, OFFSET MBOX_OD5
01FB  B8 ---- E                      MOV     AX, SEG MBOX_OD5
01FE  8E C0                          MOV     ES, AX
0200  B9 0001                        MOV     CX, 1
0203  2B D2                          SUB     DX, DX
0205  B8 0008                        MOV     AX, SC_POST
0208  CD 20                          INT     VRTX

                             ; Update the net count of transitions recorded
                             ; Variable DIST holds accumulated count.

020A  BB 0000 E                      MOV     BX, OFFSET DIST
020D  B8 ---- E                      MOV     AX, SEG DIST
0210  8E C0                          MOV     ES, AX
0212  26: FF 07                      INC     WORD PTR ES:[BX]
0215  75 06                          JNZ     QUIT
0217  83 C3 02                       ADD     BX, 2
021A  26: FF 07                      INC     WORD PTR ES:[BX]

                             ; Send EOI to PIC

021D  BA 0040                QUIT:   MOV     DX, 040H
0220  B0 20                          MOV     AL, 020H
0222  EE                             OUT     DX, AL

0223  B8 0011                        MOV     AX, UI_EXIT
0226  CD 20                          INT     VRTX

                             ODOMETER        ENDP

                             ;****************************************
                             ;*                                      *
                             ;*       SIGNPOST INPUT  INITIALIZATION *
```

```
                                ;*                                      *
                                ;****************************************

 = 0030                         INT12_VCT       EQU     WORD PTR 030H

                                ; Set interrupt vector (INT 12) for
                                ; External input ( signpost input )

 0228                           SGN_INIT        PROC            NEAR

 0228  B8 0235 R                        MOV     AX, OFFSET SGN_POST
 022B  A3 0030                          MOV     DS:INT12_VCT, AX
 022E  B8 ---- R                        MOV     AX, SEG SGN_POST
 0231  A3 0032                          MOV     DS:INT12_VCT+2, AX

 0234  C3                               RET

                                SGN_INIT        ENDP

                                ;****************************************
                                ;*                                      *
                                ;*      ISR : SIGNPOST INPUT            *
                                ;*                                      *
                                ;****************************************


 0235                           SGN_POST        PROC    FAR

 0235  50                               PUSH    AX
 0236  B8 0016                          MOV     AX, UI_ENTER
 0239  CD 20                            INT     VRTX

                                ; Post a non zero message to tell External
                                ; Input Monitoring task of this event

 023B  BB 0000 E                        MOV     BX, OFFSET SIGN_BOX
 023E  B8 ---- E                        MOV     AX, SEG SIGN_BOX
 0241  8E C0                            MOV     ES, AX
 0243  B9 0001                          MOV     CX, 1
 0246  2B D2                            SUB     DX, DX
 0248  B8 0008                          MOV     AX, SC_POST
 024B  CD 20                            INT     VRTX

                                ; send EOI to 82C59A PIC

 024D  B0 20                            MOV     AL,020H
 024F  BA 0040                          MOV     DX,040H
 0252  EE                               OUT     DX,AL

                                ; System exit

 0253  B8 0011                          MOV     AX, UI_EXIT
 0256  CD 20                            INT     VRTX
```

```
                                SGN_POST        ENDP

                                ;*****************************************
                                ;*                                       *
                                ;*    REAL TIME CLOCK INITIALIZATION     *
                                ;*                                       *
                                ;*****************************************

= 00C3                          MIN_CTR         EQU     0C3H
= 00C4                          HR_CTR          EQU     0C4H
= 00D4                          STAT            EQU     0D4H
= 00D5                          GO_RTC          EQU     0D5H

                                ; For the time being clock is being set to
                                ; 10:16 pm

0258                            RTC     PROC    NEAR


0258  BA 00C3                           MOV     DX, MIN_CTR
025B  B0 10                             MOV     AL, 010H
025D  EE                                OUT     DX, AL
025E  42                                INC     DX
025F  B0 16                             MOV     AL, 016H
0261  EE                                OUT     DX, AL

                                ; Any write operation on GO register
                                ; will start the clock

0262  E6 D5                             OUT     GO_RTC, AL
0264  C3                                RET

                                RTC     ENDP

                                ;*****************************************
                                ;*                                       *
                                ;*      ISR : SENSOR MANAGEMENT          *
                                ;*                                       *
                                ;*****************************************

                                ; There are four pairs of infrared Receive
                                ; and Transmit sensors. The transmitters and
                                ; receivers are aligned with each other
                                ; such that a beam is maintained between
                                ; them in power up stage. These get broken
                                ; whenever an object crosses them.
                                ; Each break and make of any given beam
                                ; generates an interrupt. Thus there are
                                ; 16 different interrupts, two per
                                ; beam, generated by this sensor system.
                                ;
                                ; All these interrupts are combined into one
                                ; for ease of processing and beam status is
                                ; read any time an interrupt occurs.
```

```
                                        ; This 8 bit status is EX-ORed with last
                                        ; recorded status to determine whether
                                        ; the present interrupt corresponds
                                        ; to make or break and on which sensor(s).

                                        ; NOTE : This ISR code is not completely tested

 = 0081                                          PORT_1  EQU     081H
 = 0001                                          MASK    EQU     01H

 0265                                   SENSOR           PROC            FAR

 0265                                   GET_DATA:
 0265  50                                        PUSH    AX
 0266  53                                        PUSH    BX
 0267  51                                        PUSH    CX
 0268  52                                        PUSH    DX
 0269  57                                        PUSH    DI
 026A  56                                        PUSH    SI

 026B  B8 0016                                   MOV     AX, UI_ENTER
 026E  CD 20                                     INT     VRTX

 0270  BA 0081                                   MOV     DX, PORT_1
 0273  EC                                        IN      AL, DX
 0274  8A E0                                     MOV     AH, AL
 0276  BB 0000 E                                 MOV     BX, OFFSET LAST_WD
 0279  32 07                                     XOR     AL, BYTE PTR [BX]
 027B  74 E8                                     JZ      SHORT GET_DATA
 027D  88 27                                     MOV     BYTE PTR [BX], AH

 027F  B9 0008                                   MOV     CX, 8
 0282  8B F0                                     MOV     SI, AX
 0284  81 E6 00FF                                AND     SI, 00FFH
 0288  BF 0001                                   MOV     DI, MASK
 028B  B0 01                                     MOV     AL, MASK
 028D  BB 0000 E                                 MOV     BX, OFFSET B_STAT

 0290  85 F7                            CHECK:   TEST    SI, DI
 0292  74 0B                                     JZ      RPT
 0294  84 E0                                     TEST    AH, AL
 0296  75 05                                     JNZ     BREAK
 0298  08 47 01                                  OR      BYTE PTR [BX+1], AL
 029B  EB 02                                     JMP     SHORT RPT
 029D  08 07                            BREAK:   OR      BYTE PTR [BX], AL

 029F  D1 E7                            RPT:     SHL     DI, 1
 02A1  D0 E0                                     SHL     AL, 1
 02A3  E0 EB                                     LOOPNZ  CHECK

                                        ; Now start posting to appropriate mailboxes

                                        ; First get the beam status word
```

```
02A5  8B 37                           MOV     SI, [BX]
02A7  BB 0000 E                       MOV     BX, OFFSET MBOX_E1
02AA  B8 ---- E                       MOV     AX, SEG MBOX_E1
02AD  53                              PUSH    BX
02AE  50                              PUSH    AX
02AF  8B EC                           MOV     BP, SP

02B1  B9 0010                         MOV     CX, 16
02B4  BF 0001                         MOV     DI, MASK
02B7  85 F7                   DO:     TEST    SI, DI
02B9  74 11                           JZ      SHORT DONT
02BB  51                              PUSH    CX
02BC  8B 5E 02                        MOV     BX, [BP+2]
02BF  8E 46 00                        MOV     ES, [BP]
02C2  B9 0001                         MOV     CX, 1
02C5  2B D2                           SUB     DX, DX
02C7  B8 0008                         MOV     AX, SC_POST
02CA  CD 20                           INT     VRTX

                              ; Point to next mailbox

02CC  83 C3 04                DONT:   ADD     BX, 4
02CF  89 5E 02                        MOV     [BP+2], BX
02D2  D1 E7                           SHL     DI, 1
02D4  59                              POP     CX
02D5  E0 E0                           LOOPNZ  DO

                              ; Send EOI TO 8259A

02D7  BA 0040                         MOV     DX, 040H
02DA  B0 20                           MOV     AL, 020H
02DC  EE                              OUT     DX, AL

                              ; Restore registers

02DD  5E                              POP     SI
02DE  5F                              POP     DI
02DF  5A                              POP     DX
02E0  59                              POP     CX
02E1  5B                              POP     BX

                              ; Exit system mode

02E2  B8 0011                         MOV     AX, UI_EXIT
02E5  CD 20                           INT     VRTX

                              SENSOR          ENDP

                                      ; END OF BOARD SUPPORT PACKAGE

02E7                          BSP_CODE        ENDS

                                      END
```

```
Microsoft (R) Macro Assembler  Version 4.00              7/7/87 13:40:53
                                                          Symbols-1

Segments and Groups:

                N a m e                  Size    Align   Combine Class

BSP_CODE . . . . . . . . . . . . .       02E7    PARA    PUBLIC  'BSP_CO'

Symbols:

                N a m e                  Type    Value   Attr

BREAK  . . . . . . . . . . . . . .       L NEAR  029D    BSP_CODE
BRSR . . . . . . . . . . . . . . .       Number  0003
BRSR2  . . . . . . . . . . . . . .       Number  0023
B_STAT . . . . . . . . . . . . . .       V WORD  0000            External

CFTBL_PTR_OFF  . . . . . . . . . .       WORD    0184
CFTBL_PTR_SEG  . . . . . . . . . .       WORD    0186
CHECK  . . . . . . . . . . . . . .       L NEAR  0290    BSP_CODE
CIN  . . . . . . . . . . . . . . .       F PROC  00F6    BSP_CODE        Length = 001E
CIO_RX . . . . . . . . . . . . . .       F PROC  0080    BSP_CODE        Length = 001E
CIO_TX . . . . . . . . . . . . . .       F PROC  009E    BSP_CODE        Length = 0022
CLOCK  . . . . . . . . . . . . . .       F PROC  0194    BSP_CODE        Length = 0016
COUT . . . . . . . . . . . . . . .       F PROC  0114    BSP_CODE        Length = 0022
CTR_0  . . . . . . . . . . . . . .       Number  0060
CTR_1  . . . . . . . . . . . . . .       Number  0061
CTR_2  . . . . . . . . . . . . . .       Number  0062

DIST . . . . . . . . . . . . . . .       V DWORD 0000            External
DO . . . . . . . . . . . . . . . .       L NEAR  02B7    BSP_CODE
DONT . . . . . . . . . . . . . . .       L NEAR  02CC    BSP_CODE

ENTRY  . . . . . . . . . . . . . .       L NEAR  0000    BSP_CODE
ERROR  . . . . . . . . . . . . . .       L NEAR  0027    BSP_CODE
ER_BUF . . . . . . . . . . . . . .       Number  0007
ER_COM . . . . . . . . . . . . . .       Number  0021
ER_CVT . . . . . . . . . . . . . .       Number  0020
ER_INI . . . . . . . . . . . . . .       Number  000F
ER_ISC . . . . . . . . . . . . . .       Number  0009
ER_MEM . . . . . . . . . . . . . .       Number  0003
ER_MIU . . . . . . . . . . . . . .       Number  0005
ER_NCP . . . . . . . . . . . . . .       Number  0010
ER_NMB . . . . . . . . . . . . . .       Number  0004
ER_NMP . . . . . . . . . . . . . .       Number  000B
ER_OPC . . . . . . . . . . . . . .       Number  0022
ER_PID . . . . . . . . . . . . . .       Number  000E
ER_QFL . . . . . . . . . . . . . .       Number  000D
ER_QID . . . . . . . . . . . . . .       Number  000C
ER_TCB . . . . . . . . . . . . . .       Number  0002
ER_TID . . . . . . . . . . . . . .       Number  0001
ER_TMO . . . . . . . . . . . . . .       Number  000A
ER_WTC . . . . . . . . . . . . . .       Number  0008
ER_ZMW . . . . . . . . . . . . . .       Number  0006
EXT  . . . . . . . . . . . . . . .       L NEAR  00B4    BSP_CODE
EXT2 . . . . . . . . . . . . . . .       L NEAR  012A    BSP_CODE
```

```
GET_DATA . . . . . . . . . . . . . .      L NEAR  0265    BSP_CODE
GO_RTC . . . . . . . . . . . . . . .      Number  00D5

HR_CIR . . . . . . . . . . . . . . .      Number  00C4

ICW1 . . . . . . . . . . . . . . . .      Number  0040
ICW2 . . . . . . . . . . . . . . . .      Number  0041
ICW3 . . . . . . . . . . . . . . . .      Number  0041
ICW4 . . . . . . . . . . . . . . . .      Number  0041
INIT . . . . . . . . . . . . . . . .      F PROC  0000    BSP_CODE        Length = 0051
INT09_VCT  . . . . . . . . . . . . .      WORD    0024
INT10_VCT  . . . . . . . . . . . . .      WORD    0028
INT11_VCT  . . . . . . . . . . . . .      WORD    002C
INT12_VCT  . . . . . . . . . . . . .      WORD    0030
INT14_VCT  . . . . . . . . . . . . .      WORD    0038
INT15_VCT  . . . . . . . . . . . . .      WORD    003C
INT32_VCT  . . . . . . . . . . . . .      WORD    0080
INT8_VCT . . . . . . . . . . . . . .      WORD    0020

LAST_WD  . . . . . . . . . . . . . .      V WORD  0000            External

MAIN . . . . . . . . . . . . . . . .      L FAR   0000            External
MASK . . . . . . . . . . . . . . . .      Number  0001
MBOX_E1  . . . . . . . . . . . . . .      V DWORD 0000            External
MBOX_E10 . . . . . . . . . . . . . .      V DWORD 0000            External
MBOX_E11 . . . . . . . . . . . . . .      V DWORD 0000            External
MBOX_E12 . . . . . . . . . . . . . .      V DWORD 0000            External
MBOX_E13 . . . . . . . . . . . . . .      V DWORD 0000            External
MBOX_E14 . . . . . . . . . . . . . .      V DWORD 0000            External
MBOX_E15 . . . . . . . . . . . . . .      V DWORD 0000            External
MBOX_E16 . . . . . . . . . . . . . .      V DWORD 0000            External
MBOX_E2  . . . . . . . . . . . . . .      V DWORD 0000            External
MBOX_E3  . . . . . . . . . . . . . .      V DWORD 0000            External
MBOX_E4  . . . . . . . . . . . . . .      V DWORD 0000            External
MBOX_E5  . . . . . . . . . . . . . .      V DWORD 0000            External
MBOX_E6  . . . . . . . . . . . . . .      V DWORD 0000            External
MBOX_E7  . . . . . . . . . . . . . .      V DWORD 0000            External
MBOX_E8  . . . . . . . . . . . . . .      V DWORD 0000            External
MBOX_E9  . . . . . . . . . . . . . .      V DWORD 0000            External
MBOX_OD1 . . . . . . . . . . . . . .      V DWORD 0000            External
MBOX_OD2 . . . . . . . . . . . . . .      V DWORD 0000            External
MBOX_OD3 . . . . . . . . . . . . . .      V DWORD 0000            External
MBOX_OD4 . . . . . . . . . . . . . .      V DWORD 0000            External
MBOX_OD5 . . . . . . . . . . . . . .      V DWORD 0000            External
MCR_ . . . . . . . . . . . . . . . .      Number  0002
MCR2 . . . . . . . . . . . . . . . .      Number  0022
MIN_CIR  . . . . . . . . . . . . . .      Number  00C3

OCW1 . . . . . . . . . . . . . . . .      Number  0041
OCW2 . . . . . . . . . . . . . . . .      Number  0040
OCW3 . . . . . . . . . . . . . . . .      Number  0040
ODOMETER . . . . . . . . . . . . . .      F PROC  01AA    BSP_CODE        Length = 007E
```

```
PIC_INIT . . . . . . . . . . . . . .    N PROC   013D    BSP_CODE        Length = 0011
PIT_INIT . . . . . . . . . . . . . .    N PROC   014E    BSP_CODE        Length = 0046
PIT_WCR  . . . . . . . . . . . . . .    Number   0063
PORT_1 . . . . . . . . . . . . . . .    Number   0081
P_INIT . . . . . . . . . . . . . . .    L NEAR   0029    BSP_CODE

QUIT . . . . . . . . . . . . . . . .    L NEAR   021D    BSP_CODE

RET_OK . . . . . . . . . . . . . . .    Number   0000
RPT  . . . . . . . . . . . . . . . .    L NEAR   029F    BSP_CODE
RTC  . . . . . . . . . . . . . . . .    N PROC   0258    BSP_CODE        Length = 000D

SC_ACCEPT  . . . . . . . . . . . . .    Number   0025
SC_GBLOCK  . . . . . . . . . . . . .    Number   0006
SC_GETC  . . . . . . . . . . . . . .    Number   000D
SC_GTIME . . . . . . . . . . . . . .    Number   000A
SC_LOCK  . . . . . . . . . . . . . .    Number   0020
SC_PCREATE . . . . . . . . . . . . .    Number   0022
SC_PEND  . . . . . . . . . . . . . .    Number   0009
SC_PEXTEND . . . . . . . . . . . . .    Number   0023
SC_POST  . . . . . . . . . . . . . .    Number   0008
SC_PUTC  . . . . . . . . . . . . . .    Number   000E
SC_QACCEPT . . . . . . . . . . . . .    Number   0028
SC_QCREATE . . . . . . . . . . . . .    Number   0029
SC_QINQUIRY  . . . . . . . . . . . .    Number   002A
SC_QPEND . . . . . . . . . . . . . .    Number   0027
SC_QPOST . . . . . . . . . . . . . .    Number   0026
SC_RBLOCK  . . . . . . . . . . . . .    Number   0007
SC_STIME . . . . . . . . . . . . . .    Number   000B
SC_TCREATE . . . . . . . . . . . . .    Number   0000
SC_TDELAY  . . . . . . . . . . . . .    Number   000C
SC_TDELETE . . . . . . . . . . . . .    Number   0001
SC_TINQUIRY  . . . . . . . . . . . .    Number   0005
SC_TPRIORITY . . . . . . . . . . . .    Number   0004
SC_TRESUME . . . . . . . . . . . . .    Number   0003
SC_TSLICE  . . . . . . . . . . . . .    Number   0015
SC_TSUSPEND  . . . . . . . . . . . .    Number   0002
SC_UNLOCK  . . . . . . . . . . . . .    Number   0021
SC_WAITC . . . . . . . . . . . . . .    Number   000F
SENSOR . . . . . . . . . . . . . . .    F PROC   0265    BSP_CODE        Length = 0082
SGN_INIT . . . . . . . . . . . . . .    N PROC   0228    BSP_CODE        Length = 000D
SGN_POST . . . . . . . . . . . . . .    F PROC   0235    BSP_CODE        Length = 0023
SIGN_BOX . . . . . . . . . . . . . .    V DWORD  0000            External
STAT . . . . . . . . . . . . . . . .    Number   00D4

TBL  . . . . . . . . . . . . . . . .    V BYTE   0000            External
TXRDY  . . . . . . . . . . . . . . .    F PROC   00C0    BSP_CODE        Global  Length = 0007
TX_RDY . . . . . . . . . . . . . . .    F PROC   0136    BSP_CODE        Global  Length = 0007

UCR  . . . . . . . . . . . . . . . .    Number   0001
UCR2 . . . . . . . . . . . . . . . .    Number   0021
UDATA2 . . . . . . . . . . . . . . .    Number   0020
UI_ENTER . . . . . . . . . . . . . .    Number   0016
UI_EXIT  . . . . . . . . . . . . . .    Number   0011
```

```
UI_RXCHR . . . . . . . . . . . . . .       Number  0013
UI_TIMER . . . . . . . . . . . . . .       Number  0012
UI_TXRDY . . . . . . . . . . . . . .       Number  0014
USART2 . . . . . . . . . . . . . . .       N PROC  00C7    BSP_CODE        Length = 002F
USART_DATA . . . . . . . . . . . . .       Number  0000
USART_INIT . . . . . . . . . . . . .       N PROC  0051    BSP_CODE        Length = 002F

VCS  . . . . . . . . . . . . . . . .       Number  0A00
VIP  . . . . . . . . . . . . . . . .       Number  0000
VRTX . . . . . . . . . . . . . . . .       WORD    0020
VRTX_GO  . . . . . . . . . . . . . .       Number  0031
VRTX_INIT  . . . . . . . . . . . . .       Number  0030

   1083 Source  Lines
   1085 Total   Lines
    170 Symbols

  45350 Bytes symbol space free

      0 Warning Errors
      0 Severe  Errors
```

```
                                ;****************************************
                                ;*                                      *
                                ;*   TEST.ASM : Test Program for BSP    *
                                ;*                                      *
                                ;*   The tasks included here are part   *
                                ;*   of the application software        *
                                ;*   developed for multitasking VDAS    *
                                ;*   They are presented here solely     *
                                ;*   for the purpose of illustration    *
                                ;*   and to present a sample program    *
                                ;*   to test Board Support Package.     *
                                ;                                       *
                                ;****************************************

                                ; TXRDY routine in BSP module

                                        EXTRN   TXRDY:FAR

                                ; Include VRTX system call equates
                            C           include vrtx.inc
                            C
                            C
                            C   ; Please see previous module listings for
                            C   ; for VRTX system call equates
                            C   ; These are suppressed here for brevity.
                            C
                            C   .LIST
```

```
                                    page
                                    ; Include Constant Definitions
                                  C         include const.inc
                                  C ;*****************************************
                                  C ;                                       *
                                  C ;         CONSTANTS' DEFINITIONS        *
                                  C ;                                       *
                                  C ;*****************************************
                                  C
                                  C ; constant definitions
                                  C
 = 000D                           C CR        EQU     0DH     ; carriage return
 = 000A                           C LF        EQU     0AH     ; line feed
 = 0000                           C EOS       EQU     00H     ; end of string marker
                                  C
 = 00C2                           C SEC_CTR EQU     0C2H    ; seconds counter
 = 00C3                           C MIN_CTR EQU     0C3H    ; minutes counter
 = 00C4                           C HR_CTR  EQU     0C4H    ; hours counter
 = 00D4                           C STAT      EQU     0D4H    ; RTC status
 = 00D5                           C GO_RTC  EQU     0D5H    ; RTC start port
                                  C
                                  C ; One minute corresponds to 6000 ticks
                                  C ; 1 min = 6000 * 1 tick (10 ms) = 60 sec
                                  C
 = 1770                           C ONE_MIN           EQU     6000D
 = 2EE0                           C TWO_MIN           EQU     12000D
                                  C
                                  C ; Log Types are assigned here
                                  C
 = 0001                           C LOG1              EQU     1
 = 0002                           C LOG2              EQU     2
 = 0003                           C LOG3              EQU     3
 = 0004                           C LOG4              EQU     4
 = 0005                           C LOG5              EQU     5
 = 0006                           C LOG6              EQU     6
 = 0007                           C LOG7              EQU     7
 = 0008                           C LOG8              EQU     8
 = 0009                           C LOG9              EQU     9
 = 000A                           C LOG10             EQU     10
 = 000B                           C LOG11             EQU     11
 = 000C                           C LOG12             EQU     12
                                  C
```

```
                                      page
                                      ; Include Configuration Table
                                   C          include cftbl.inc
                                   C  ;*****************************************
                                   C  ;                                        *
                                   C  ;          CONFIGURATION TABLE           *
                                   C  ;                                        *
                                   C  ;*****************************************
                                   C
 0000                              C  CFTBL            SEGMENT  PARA   'DATA'
                                   C
                                   C          PUBLIC  TBL
 0000                              C  TBL     LABEL   BYTE
                                   C
 0000  0200                        C          DW      0200H   ; vrtx workspace starts at 02000H
 0002  0200                        C          DW      0200H   ; size = 16 * 512 bytes (8k)
 0004  0000                        C          DW      0       ; reserved = 0
 0006  0000                        C          DW      0       ; no separate ISR stack (=0)
 0008  0000                        C          DW      0       ; reserved =0
 000A  0000                        C          DW      0       ; reserved =0
 000C  0000                        C          DW      0       ; reserved =0
 000E  0010                        C          DW      0010H   ; user stack size = 16 * 16 bytes
 0010  00 00 00 00                 C          DD      0       ; reserved =0
 0014  000A                        C          DW      0010    ; user task count = 10
 0016  0000                        C          DW      0       ; reserved =0
 0018  0000 ---- E                 C          DD      TXRDY   ; TXRDY transmit driver address
 001C  00 00 00 00                 C          DD      0       ; no tcreate routine
 0020  00 00 00 00                 C          DD      0       ; no tdelete routine
 0024  00 00 00 00                 C          DD      0       ; no tswitch routine
 0028  00 00 00 00                 C          DD      0       ; no other component only VRTX
                                   C
 002C                              C  CFTBL   ENDS
                                   C
```

```
                                     page
0000                                 TEST_CODE      SEGMENT PARA    'CODE'
                                              ASSUME CS:TEST_CODE, DS:TEST_DATA
                                     ; The parent process - creates tasks
                                   C          include main.tsk
                                   C ;****************************************
                                   C ;                                       *
                                   C ;        Code for task "MAIN"           *
                                   C ;                                       *
                                   C ;****************************************
                                   C
                                   C          PUBLIC  MAIN
                                   C
0000                               C MAIN     PROC    FAR
                                   C
                                   C ; set correct data segment
                                   C
0000  B8 ---- R                    C          MOV     AX, TEST_DATA
0003  8E D8                        C          MOV     DS, AX
                                   C
                                   C ; print ("I/O test started\n")
                                   C
0005  BB 0107 R                    C          MOV     BX, OFFSET MSG1
0008  E8 0486 R                    C          CALL    PUTS
                                   C
                                   C ; delay task for 1000/2 ticks
                                   C
000B  B9 01F4                      C          MOV     CX, 500
000E  BA 0000                      C          MOV     DX, 0
0011  B8 000C                      C          MOV     AX, SC_TDELAY
0014  CD 20                        C          INT     VRTX
                                   C
                                   C ; create task to record power on log
                                   C ; id = 1 pri = 1
                                   C
0016  BB 00B7 R                    C          MOV     BX, OFFSET PWR_ON
0019  B8 ---- R                    C          MOV     AX, SEG PWR_ON
001C  8E C0                        C          MOV     ES, AX
001E  B1 01                        C          MOV     CL, 1          ; id = 1
0020  B5 01                        C          MOV     CH, 1          ; TOP PRIORITY
0022  B8 0000                      C          MOV     AX, SC_TCREATE
0025  CD 20                        C          INT     VRTX
                                   C
                                   C ; create task to print ("MULTITASKING..")
                                   C ; id = 2 pri = 6
                                   C
0027  BB 00F8 R                    C          MOV     BX, OFFSET MESSAGE
002A  B8 ---- R                    C          MOV     AX, SEG MESSAGE
002D  8E C0                        C          MOV     ES, AX
002F  B1 02                        C          MOV     CL, 2
```

```
0031  B5 06                   C           MOV     CH, 6
0033  B8 0000                 C           MOV     AX, SC_TCREATE
0036  CD 20                   C           INT     VRTX
                              C
                              C   ; create task to wait for control C
                              C   ; id = 3  pri = 4
                              C
0038  BB 012B R               C           MOV     BX, OFFSET CTLC
003B  B8 ---- R               C           MOV     AX, SEG CTLC
003E  8E C0                   C           MOV     ES, AX
0040  B1 03                   C           MOV     CL, 3
0042  B5 04                   C           MOV     CH, 4
0044  B8 0000                 C           MOV     AX, SC_TCREATE
0047  CD 20                   C           INT     VRTX
                              C
                              C   ; create a task  to output Real Time
                              C   ; id = 4  pri = 5
                              C
0049  BB 017E R               C           MOV     BX, OFFSET RTASK
004C  B8 ---- R               C           MOV     AX, SEG RTASK
004F  8E C0                   C           MOV     ES, AX
0051  B1 04                   C           MOV     CL, 4
0053  B5 05                   C           MOV     CH, 5
0055  B8 0000                 C           MOV     AX, SC_TCREATE
0058  CD 20                   C           INT     VRTX
                              C
                              C   ; create a task for one_min_idle
                              C   ; id = 5  pri = 3
                              C
005A  BB 01DF R               C           MOV     BX, OFFSET IDLE_1M
005D  B8 ---- R               C           MOV     AX, SEG IDLE_1M
0060  8E C0                   C           MOV     ES, AX
0062  B1 05                   C           MOV     CL, 5
0064  B5 03                   C           MOV     CH, 3
0066  B8 0000                 C           MOV     AX, SC_TCREATE
0069  CD 20                   C           INT     VRTX
                              C
                              C   ; create a task to write one min idle log
                              C   ; id = 6  pri = 4
                              C
006B  BB 0298 R               C           MOV     BX, OFFSET IDLE1
006E  B8 ---- R               C           MOV     AX, SEG IDLE1
0071  8E C0                   C           MOV     ES, AX
0073  B1 06                   C           MOV     CL, 6
0075  B5 04                   C           MOV     CH, 4
0077  B8 0000                 C           MOV     AX, SC_TCREATE
007A  CD 20                   C           INT     VRTX
                              C
                              C   ; create a task for two_min_idle
                              C   ; id = 7  pri = 3
                              C
007C  BB 022A R               C           MOV     BX, OFFSET IDLE_2M
007F  B8 ---- R               C           MOV     AX, SEG IDLE_2M
0082  8E C0                   C           MOV     ES, AX
```

```
0084  B1 07                C          MOV     CL, 7
0086  B5 03                C          MOV     CH, 3
0088  B8 0000              C          MOV     AX, SC_TCREATE
008B  CD 20                C          INT     VRTX
                           C
                           C  ; create a task to write 2 min idle log
                           C  ; id = 8  pri = 4
                           C
008D  BB 02E8 R            C          MOV     BX, OFFSET IDLE2
0090  B8 ---- R            C          MOV     AX, SEG IDLE2
0093  8E C0                C          MOV     ES, AX
0095  B1 08                C          MOV     CL, 8
0097  B5 04                C          MOV     CH, 4
0099  B8 0000              C          MOV     AX, SC_TCREATE
009C  CD 20                C          INT     VRTX
                           C
                           C  ; create a task to monitor external input
                           C  ; id = 9  pri = 3
                           C
009E  BB 033F R            C          MOV     BX, OFFSET EXT_IP
00A1  B8 ---- R            C          MOV     AX, SEG EXT_IP
00A4  8E C0                C          MOV     ES, AX
00A6  B1 09                C          MOV     CL, 9
00A8  B5 03                C          MOV     CH, 3
00AA  B8 0000              C          MOV     AX, SC_TCREATE
00AD  CD 20                C          INT     VRTX
                           C
                           C                            ; delete self
                           C
00AF  B9 0000              C          MOV     CX, 0
00B2  B8 0001              C          MOV     AX, SC_TDELETE
00B5  CD 20                C          INT     VRTX
                           C
                           C  MAIN    ENDP
```

```
                                page
                                ;****************************************
                                ;                                       *
                                ;       Code for task POWER ON LOG      *
                                ;                                       *
                                ;****************************************

                                ; This tasks generates and stores a 5 byte
                                ; log on power up. Here it does so when
                                ; executed after reset.

 00B7                           PWR_ON  PROC    NEAR

 00B7  B9 0004                          MOV     CX, 4
 00BA  BB 01EC R                        MOV     BX, OFFSET LOG_BUF
 00BD  C6 07 01                         MOV     BYTE PTR [BX], LOG1
 00C0  43                       ZERO:   INC     BX
 00C1  C6 07 00                         MOV     BYTE PTR [BX], 0
 00C4  E0 FA                            LOOPNZ  ZERO

 00C6  E8 0498 R                        CALL     LOG_WRITE

                                ; wait for key to crt resource

                                ; Setup mailbox context and pend indefinitely

 00C9  BB 0000 R                        MOV     BX, OFFSET CRT
 00CC  B8 ---- R                        MOV     AX, SEG CRT
 00CF  8E C0                            MOV     ES, AX
 00D1  2B C9                            SUB     CX, CX
 00D3  2B D2                            SUB     DX, DX
 00D5  B8 0009                          MOV     AX, SC_PEND
 00D8  CD 20                            INT     VRTX

                                ; print ("The power on log created\n")
                                ; point to string then call print routine

 00DA  BB 016D R                        MOV     BX, OFFSET MSG5
 00DD  E8 0486 R                        CALL    PUTS

                                ; return crt resource key to lock

 00E0  BB 0000 R                        MOV     BX, OFFSET CRT
 00E3  B8 ---- R                        MOV     AX, SEG CRT
 00E6  8E C0                            MOV     ES, AX
 00E8  B9 0001                          MOV     CX, 1
 00EB  B8 0008                          MOV     AX, SC_POST
 00EE  CD 20                            INT     VRTX

                                ; delete itself

 00F0  B9 0000                          MOV     CX, 0
 00F3  B8 0001                          MOV     AX, SC_TDELETE
 00F6  CD 20                       INT     VRTX

                          PWR_ON  ENDP

                          ;*****************************************
                          ;                                       *
                          ;       Code for task 'MESSAGE"         *
                          ;                                       *
                          ;Reproduced from the example in         *
                          ;BSP Manual [9] from Hunter and Ready   *
                          ;                                       *
                          ;*****************************************

 00F8                     MESSAGE          PROC    NEAR

                          ; wait for key to crt resource

                          ; Setup CRT mailbox context and pend indefinite
                          ly

 00F8  BB 0000 R          PRMSG:  MOV     BX, OFFSET CRT
 00FB  B8 ---- R                  MOV     AX, SEG CRT
 00FE  8E C0                      MOV     ES, AX
 0100  2B C9                      SUB     CX, CX
 0102  2B D2                      SUB     DX, DX
 0104  B8 0009                    MOV     AX, SC_PEND
 0107  CD 20                      INT     VRTX

                          ; print ("multitasking in action..\n")

 0109  BB 0121 R                  MOV     BX, OFFSET MSG2
 010C  E8 0486 R                  CALL    PUTS

                          ; return crt resource key to lock

 010F  BB 0000 R                  MOV     BX, OFFSET CRT
 0112  B8 ---- R                  MOV     AX, SEG CRT
 0115  8E C0                      MOV     ES, AX
 0117  B9 0001                    MOV     CX, 1
 011A  B8 0008                    MOV     AX, SC_POST
 011D  CD 20                      INT     VRTX

                          ; wait for 1 sec before reprinting

 011F  B9 0064                    MOV     CX, 100
 0122  2B D2                      SUB     DX, DX
 0124  B8 000C                    MOV     AX, SC_TDELAY
 0127  CD 20                      INT     VRTX

 0129  EB CD                      JMP     PRMSG

                          MESSAGE ENDP
```

```
                                        ;****************************************
                                        ;                                      *
                                        ;       Code for task "CTLC"           *
                                        ;                                      *
                                        ;Reproduced from the example in        *
                                        ;BSP Manual [9] from Hunter and Ready   *
                                        ;                                      *
                                        ;****************************************

 012B                                   CTLC    PROC    NEAR

                                        ; wait for a control C

 012B  B5 03                            WAITC:  MOV     CH,03H          ; ascii code
 012D  B8 000F                                  MOV     AX, SC_WAITC    ; for Cntl-C
 0130  CD 20                                    INT     VRTX

                                        ; wait for key to crt resource

 0132  BB 0000 R                                MOV     BX, OFFSET CRT
 0135  B8 ---- R                                MOV     AX, SEG CRT
 0138  8E C0                                    MOV     ES, AX
 013A  2B C9                                    SUB     CX, CX
 013C  2B D2                                    SUB     DX, DX
 013E  B8 0009                                  MOV     AX, SC_PEND
 0141  CD 20                                    INT     VRTX

                                        ; print ("\ncontrol C received...\n")
                                        ; print ("=>")

                                        ; point to message string and print

 0143  BB 013C R                                MOV     BX, OFFSET MSG3
 0146  E8 0486 R                                CALL    PUTS

                                        ; point to prompt string and print

 0149  BB 016A R                                MOV     BX, OFFSET MSG4
 014C  E8 0486 R                                CALL    PUTS

                                        ; input a line of text from crt
                                        ; echo the line to the crt

                                        ; point to the input buffer before
                                        ; getting the string. Also echo it

 014F  BB 0004 R                                MOV     BX, OFFSET BUF
 0152  E8 0464 R                                CALL    GETS
 0155  BB 0004 R                                MOV     BX, OFFSET BUF
 0158  E8 0486 R                                CALL    PUTS

                                        ; Echo CR LF followed by string
```

```
015B  BB 0104 R                        MOV     BX, OFFSET NL
015E  E8 0486 R                        CALL    PUTS

                              ; delay task for 1000 ticks

0161  B9 03E8                          MOV     CX, 1000              ; ticks = 1000
0164  BA 0000                          MOV     DX, 0
0167  B8 000C                          MOV     AX, SC_TDELAY
016A  CD 20                            INT     VRTX

                              ; return crt resource key to lock

016C  BB 0000 R                        MOV     BX, OFFSET CRT
016F  B8 ---- R                        MOV     AX, SEG CRT
0172  8E C0                            MOV     ES, AX
0174  B9 0001                          MOV     CX, 1
0177  B8 0008                          MOV     AX, SC_POST
017A  CD 20                            INT     VRTX

                              ; Continue looking for Control-C

017C  EB AD                            JMP     WAITC

                              CTLC     ENDP

                              ;****************************************
                              ;*                                      *
                              ;*   RTASK    PRINTS TIME ON REQUEST    *
                              ;*                                      *
                              ;****************************************

017E                          RTASK            PROC    NEAR

017E  B8 000D                 RT:      MOV     AX, SC_GETC
0181  CD 20                            INT     VRTX
0183  80 FD 52                         CMP     CH, 'R'
0186  75 F6                            JNE     RT
0188  B8 000E                          MOV     AX, SC_PUTC
018B  CD 20                            INT     VRTX

                              ; Get hours, minutes and seconds from RTC
                              ; convet them into ASCII and store them
                              ; in display buffer for outputting

018D  BB 0213 R                        MOV     BX, OFFSET REAL_TIME
0190  E8 042E R                        CALL    GET_HR
0193  E8 03F4 R                        CALL    BCD_ASC
0196  E8 0413 R                        CALL    GET_MS
0199  50                               PUSH    AX
019A  8A E0                            MOV     AH, AL
019C  E8 03F4 R                        CALL    BCD_ASC
019F  58                               POP     AX
01A0  E8 03F4 R                        CALL    BCD_ASC
```

```
                                ; now store cr,lf and eos
                                ; delete extra ':'

01A3  4B                                DEC     BX
01A4  B8 0D0A                           MOV     AX, 0D0AH
01A7  89 07                             MOV     [BX], AX
01A9  43                                INC     BX
01AA  43                                INC     BX
01AB  C6 07 00                          MOV     BYTE PTR [BX], EOS

                                ; look for crt key

01AE  BB 0000 R                         MOV     BX, OFFSET CRT
01B1  B8 ---- R                         MOV     AX, SEG CRT
01B4  8E C0                             MOV     ES, AX
01B6  2B C9                             SUB     CX, CX
01B8  2B D2                             SUB     DX, DX
01BA  B8 0009                           MOV     AX, SC_PEND
01BD  CD 20                             INT     VRTX

01BF  BB 022C R                         MOV     BX, OFFSET RT_MSG
01C2  E8 0486 R                         CALL    PUTS
01C5  BB 0213 R                         MOV     BX, OFFSET REAL_TIME
01C8  E8 0486 R                         CALL    PUTS

                                ; return crt key

01CB  BB 0000 R                         MOV     BX, OFFSET CRT
01CE  B8 ---- R                         MOV     AX, SEG CRT
01D1  8E C0                             MOV     ES, AX
01D3  B9 0001                           MOV     CX, 1
01D6  2B D2                             SUB     DX, DX
01D8  B8 0008                           MOV     AX, SC_POST
01DB  CD 20                             INT     VRTX


                                ; Look for new request for time

01DD  EB 9F                             JMP     RT

                                RTASK   ENDP

                                ;****************************************
                                ;*                                      *
                                ;*      TASK : ONE MINUTE IDLE          *
                                ;*                                      *
                                ;****************************************


01DF                            IDLE_1M         PROC            NEAR

                                ; setup mailbox context : timeout = 1min

01DF  BB 0000 R                 M1:     MOV     BX, OFFSET MBOX_OD1
```

```
01E2  B8 ---- R                        MOV     AX, SEG MBOX
01E5  8E C0                            MOV     ES, AX
01E7  B9 1770                          MOV     CX, ONE_MIN
                              ;        MOV     CX, 0
01EA  2B D2                            SUB     DX, DX
01EC  B8 0009                          MOV     AX, SC_PEND
01EF  CD 20                            INT     VRTX
01F1  3D 000A                          CMP     AX, ER_TMO
01F4  74 0F                            JE      SHORT M10

                              ; Did message arrive within a minute ?

                              ; If yes then remove msg from OD2 also
                              ; Message is removed by accepting it from
                              ; the mailbox, if it is there, and then
                              ; discarding it. No Rescheduling is done

01F6  BB 0004 R                        MOV     BX, OFFSET MBOX_OD2
01F9  B8 ---- R                        MOV     AX, SEG MBOX
01FC  8E C0                            MOV     ES, AX
01FE  B8 0025                          MOV     AX, SC_ACCEPT
0201  CD 20                            INT     VRTX

0203  EB DA                            JMP     SHORT M1

                              ; If no then post message for one_min_idle_log

0205  BB 0014 R               M10:     MOV     BX, OFFSET LOG_1MIN
0208  B8 ---- R                        MOV     AX, SEG MBOX
020B  8E C0                            MOV     ES, AX
020D  B9 0001                          MOV     CX, 01H
0210  2B D2                            SUB     DX, DX
0212  B8 0008                          MOV     AX, SC_POST
0215  CD 20                            INT     VRTX

                              ; Wait till vehicle moves

0217  BB 0004 R                        MOV     BX, OFFSET MBOX_OD2
021A  B8 ---- R                        MOV     AX, SEG MBOX
021D  8E C0                            MOV     ES, AX
021F  2B C9                            SUB     CX, CX
0221  2B D2                            SUB     DX, DX
0223  B8 0009                          MOV     AX, SC_PEND
0226  CD 20                            INT     VRTX

                              ; Start monitoring for another one min idle log

0228  EB B5                            JMP     M1

                              IDLE_1M          ENDP
```

```
                                page
                                ;****************************************
                                ;*                                      *
                                ;*  TASK : TWO OR MORE MINUTE IDLE      *
                                ;*                                      *
                                ;****************************************
                                ;
 022A                           IDLE_2M         PROC            NEAR

                                ; Get present value of VRTX maintained time
                                ; and save it <DX:CX> on the stack

 022A  B8 000A                  M2:     MOV     AX, SC_GTIME
 022D  CD 20                            INT     VRTX
 022F  52                               PUSH    DX
 0230  51                               PUSH    CX

                                ; Setup odometer mailbox context
                                ; Pend at it for two minutes

 0231  BB 0008 R                        MOV     BX, OFFSET MBOX_OD3
 0234  B8 ---- R                        MOV     AX, SEG MBOX
 0237  8E C0                            MOV     ES, AX
 0239  B9 2EE0                          MOV     CX, TWO_MIN
 023C  2B D2                            SUB     DX, DX
 023E  B8 0009                          MOV     AX, SC_PEND
 0241  CD 20                            INT     VRTX
 0243  3D 000A                          CMP     AX, ER_TMO
 0246  74 0F                            JE      SHORT M20

                                ; Did a message arrive within 2 minutes ?

                                ; If it did then remove message from OD4 also

 0248  BB 000C R                        MOV     BX, OFFSET MBOX_OD4
 024B  B8 ---- R                        MOV     AX, SEG MBOX
 024E  8E C0                            MOV     ES, AX
 0250  B8 0025                          MOV     AX, SC_ACCEPT
 0253  CD 20                            INT     VRTX

 0255  EB D3                            JMP     SHORT M2

                                ; If not then wait till vehicle moves and
                                ; then post message for two_min_idle_log

 0257  BB 000C R                M20:    MOV     BX, OFFSET MBOX_OD4
 025A  B8 ---- R                        MOV     AX, SEG MBOX
 025D  8E C0                            MOV     ES, AX
 025F  2B C9                            SUB     CX, CX
 0261  2B D2                            SUB     DX, DX
 0263  B8 0009                          MOV     AX, SC_PEND
 0266  CD 20                            INT     VRTX
```

```
                                ; Get the time when vehicle finally moves
                                ; Substract the initial time value from
                                ; this to get time interval in <DX:CX>

0268  B8 000A                           MOV     AX, SC_GTIME
026B  CD 20                             INT     VRTX
026D  5B                                POP     BX
026E  2B CB                             SUB     CX, BX
0270  5B                                POP     BX
0271  1B D3                             SBB     DX, BX

                                ; Variable HALT_TIME holds the interval for
                                ; which the vehicle was stationary

0273  BB 01E8 R                         MOV     BX, OFFSET HALT_TIME
0276  B8 ---- R                         MOV     AX, SEG HALT_TIME
0279  8E C0                             MOV     ES, AX
027B  26: 89 0F                         MOV     ES:[BX], CX
027E  83 C3 02                          ADD     BX, 2
0281  26: 89 17                         MOV     ES:[BX], DX

                                ; Post message for two_min_idle_log_write task

0284  BB 0018 R                         MOV     BX, OFFSET LOG_2MIN
0287  B8 ---- R                         MOV     AX, SEG LOG_2MIN
028A  8E C0                             MOV     ES, AX
028C  B9 0002                           MOV     CX, 02H
028F  2B D2                             SUB     DX, DX
0291  B8 0008                           MOV     AX, SC_POST
0294  CD 20                             INT     VRTX

                                ; Start monitoring for another two min idle log

0296  EB 92                             JMP     M2

                                IDLE_2M         ENDP

                                ;****************************************
                                ;*                                      *
                                ;*          WRITE_ONE_MIN_IDLE_LOG      *
                                ;*                                      *
                                ;****************************************

0298                            IDLE1           PROC            NEAR

                                ; This task pends indefinitely at the mailbox
                                ; LOG_1MIN which is posted to by IDLE_1M task
                                ; whenever that type of log is generated.

                                ; It creates a log in LOG_BUF and then calls
                                ; LOG_WRITE procedure to transfer to log tbl
```

```
0298  BB 0014 R              IDL:   MOV    BX, OFFSET LOG_1MIN
029B  B8 ---- R                     MOV    AX, SEG MBOX
029E  8E C0                         MOV    ES, AX
02A0  2B C9                         SUB    CX, CX
02A2  2B D2                         SUB    DX, DX
02A4  B8 0009                       MOV    AX, SC_PEND
02A7  CD 20                         INT    VRTX
02A9  BB 01EC R                     MOV    BX, OFFSET LOG_BUF
02AC  C6 07 02                      MOV    BYTE PTR [BX], LOG2

                             ; Get real time in minutes and seconds.

02AF  E8 0413 R                     CALL   GET_MS
02B2  43                            INC    BX
02B3  89 07                         MOV    [BX], AX
02B5  83 C3 02                      ADD    BX, 2
02B8  2B C0                         SUB    AX, AX
02BA  89 07                         MOV    [BX], AX
02BC  E8 0498 R                     CALL   LOG_WRITE

                             ; Get crt resource

02BF  BB 0000 R                     MOV    BX, OFFSET CRT
02C2  B8 ---- R                     MOV    AX, SEG CRT
02C5  8E C0                         MOV    ES, AX
02C7  2B C9                         SUB    CX, CX
02C9  2B D2                         SUB    DX, DX
02CB  B8 0009                       MOV    AX, SC_PEND
02CE  CD 20                         INT    VRTX

                             ; print 'One Minute Idle Log Created'

02D0  BB 0186 R                     MOV    BX, OFFSET MSG7
02D3  E8 0486 R                     CALL   PUTS

                             ; return crt

02D6  B9 0001                       MOV    CX, 1
02D9  BB 0000 R                     MOV    BX, OFFSET CRT
02DC  B8 ---- R                     MOV    AX, SEG CRT
02DF  8E C0                         MOV    ES, AX
02E1  B8 0008                       MOV    AX, SC_POST
02E4  CD 20                         INT    VRTX

02E6  EB B0                         JMP    IDL

                             IDLE1  ENDP

                             ;****************************************
                             ;*                                      *
                             ;*        WRITE TWO_MIN_IDLE_LOG        *
                             ;*                                      *
                             ;****************************************
```

```
 02E8                              IDLE2          PROC           NEAR

                                   ; Similar to the one above but creates and
                                   ; writes log pertaining to two or more
                                   ; minutes idle conditions

 02E8  BB 0018 R                   IDL2:   MOV    BX, OFFSET LOG_2MIN
 02EB  B8 ---- R                           MOV    AX, SEG MBOX
 02EE  8E C0                               MOV    ES, AX
 02F0  2B C9                               SUB    CX, CX
 02F2  2B D2                               SUB    DX, DX
 02F4  B8 0009                             MOV    AX, SC_PEND
 02F7  CD 20                               INT    VRTX

 02F9  BB 01EC R                           MOV    BX, OFFSET LOG_BUF
 02FC  C6 07 03                            MOV    BYTE PTR [BX], LOG3
 02FF  43                                  INC    BX
 0300  53                                  PUSH   BX

                                   ; Get the interval for which vehicle idled

 0301  BB 01E8 R                           MOV    BX, OFFSET HALT_TIME
 0304  8B 0F                               MOV    CX, [BX]
 0306  83 C3 02                            ADD    BX, 2
 0309  8B 17                               MOV    DX, [BX]
 030B  5B                                  POP    BX
 030C  89 0F                               MOV    [BX], CX
 030E  83 C3 02                            ADD    BX, 2
 0311  89 17                               MOV    [BX], DX

                                   ; Write the actual log in Log Table

 0313  E8 0498 R                           CALL   LOG_WRITE

                                   ; get crt

 0316  BB 0000 R                           MOV    BX, OFFSET CRT
 0319  B8 ---- R                           MOV    AX, SEG CRT
 031C  8E C0                               MOV    ES, AX
 031E  2B C9                               SUB    CX, CX
 0320  2B D2                               SUB    DX, DX
 0322  B8 0009                             MOV    AX, SC_PEND
 0325  CD 20                               INT    VRTX

                                   ; print 'Two Minute Idle Log Created'

 0327  BB 01A7 R                           MOV    BX, OFFSET MSG8
 032A  E8 0486 R                           CALL   PUTS

                                   ; return crt

 032D  B9 0001                             MOV    CX, 1
 0330  BB 0000 R                           MOV    BX, OFFSET CRT
 0333  B8 ---- R                           MOV    AX, SEG CRT
```

```
0336  8E C0                            MOV     ES, AX
0338  B8 0008                          MOV     AX, SC_POST
033B  CD 20                            INT     VRTX

033D  EB A9                            JMP     IDL2

                                 IDLE2   ENDP

                                 ;****************************************
                                 ;*                                      *
                                 ;*  EXT_IP : SIGNPOST INPUT TASK        *
                                 ;*                                      *
                                 ;****************************************

033F                             EXT_IP          PROC            NEAR

033F  BB 0028 R                  SGNP:   MOV     BX, OFFSET SIGN_BOX
0342  B8 ---- R                          MOV     AX, SEG SIGN_BOX
0345  8E C0                              MOV     ES, AX
0347  2B C9                              SUB     CX, CX
0349  2B D2                              SUB     DX, DX
034B  B8 0009                            MOV     AX, SC_PEND
034E  CD 20                              INT     VRTX

                                 ; Write the log type first

0350  BB 01EC R                          MOV     BX, OFFSET LOG_BUF
0353  C6 07 08                           MOV     BYTE PTR [BX], LOG8

                                 ; Get real time in mins and secs

0356  E8 0413 R                          CALL    GET_MS
0359  43                                 INC     BX
035A  89 07                              MOV     [BX], AX
035C  83 C3 02                           ADD     BX, 2

                                 ; Get incremental distance (since last log)

035F  E8 043B R                          CALL    GET_DIST
0362  88 07                              MOV     [BX], AL
0364  43                                 INC     BX
0365  C6 07 00                           MOV     BYTE PTR [BX], 0

                                 ; Write the log in the Log Table

0368  E8 0498 R                          CALL    LOG_WRITE

                                 ; get crt

036B  BB 0000 R                          MOV     BX, OFFSET CRT
036E  B8 ---- R                          MOV     AX, SEG CRT
0371  8E C0                              MOV     ES, AX
0373  2B C9                              SUB     CX, CX
0375  2B D2                              SUB     DX, DX
```

```
0377  B8 0009                  MOV     AX, SC_PEND
037A  CD 20                    INT     VRTX

                       ; print ("\n External Input Log Created \n")

037C  BB 01C8 R                MOV     BX, OFFSET MSG10
037F  E8 0486 R                CALL    PUTS

                       ; return crt

0382  BB 0000 R                MOV     BX, OFFSET CRT
0385  B8 ---- R                MOV     AX, SEG CRT
0388  8E C0                    MOV     ES, AX
038A  B8 0008                  MOV     AX, SC_POST
038D  CD 20                    INT     VRTX

                       ; wait for 3 minutes to avoid registering
                       ; switching noise
                       ; 3 mins = 18000 ticks ( 180 secs )

038F  B9 4650                  MOV     CX, 18000
0392  2B D2                    SUB     DX, DX
0394  B8 000C                  MOV     AX, SC_TDELAY
0397  CD 20                    INT     VRTX

0399  EB A4                    JMP     SHORT  SGNP

                       EXT_IP          ENDP

                       ;*****************************************
                       ;*                                       *
                       ;* ASC- CNVERTS HEX CHAR IN AX TO ASCII *
                       ;*                                       *
                       ;*****************************************

039B                   ASC     PROC    NEAR

039B  BB 0203 R                MOV     BX, OFFSET TIME
039E  8B C2                    MOV     AX, DX
03A0  E8 03B8 R                CALL    OPN
03A3  C6 07 3A                 MOV     BYTE PTR [BX], ':'
03A6  43                       INC     BX
03A7  8B C1                    MOV     AX, CX
03A9  E8 03B8 R                CALL    OPN
03AC  B8 0D0A          OVER:   MOV     AX, 0D0AH
03AF  89 07                    MOV     [BX], AX
03B1  83 C3 02                 ADD     BX, 2
03B4  C6 07 00                 MOV     BYTE PTR [BX], EOS
03B7  C3                       RET

                       ASC     ENDP

                       ;*****************************************
```

```
                                 ;*                                       *
                                 ;*              OPERATIONS               *
                                 ;*                                       *
                                 ;*****************************************

                                 ; Converts a byte into two ascii characters

 03B8                            OPN        PROC           NEAR


 03B8  51                                PUSH    CX
 03B9  8B C8                             MOV     CX, AX
 03BB  D0 C8                             ROR     AL, 1
 03BD  D0 C8                             ROR     AL, 1
 03BF  D0 C8                             ROR     AL, 1
 03C1  D0 C8                             ROR     AL, 1
 03C3  E8 03EB R                         CALL    CONV
 03C6  88 07                             MOV     [BX], AL
 03C8  43                                INC     BX
 03C9  8B C1                             MOV     AX, CX
 03CB  E8 03EB R                         CALL    CONV
 03CE  88 07                             MOV     [BX], AL
 03D0  43                                INC     BX
 03D1  8A C5                             MOV     AL, CH
 03D3  D0 C8                             ROR     AL, 1
 03D5  D0 C8                             ROR     AL, 1
 03D7  D0 C8                             ROR     AL, 1
 03D9  D0 C8                             ROR     AL, 1
 03DB  E8 03EB R                         CALL    CONV
 03DE  88 07                             MOV     [BX], AL
 03E0  43                                INC     BX
 03E1  8B C1                             MOV     AX, CX
 03E3  E8 03EB R                         CALL    CONV
 03E6  88 07                             MOV     [BX], AL
 03E8  43                                INC     BX
 03E9  59                                POP     CX
 03EA  C3                                RET

                                 OPN     ENDP

                                 ;*****************************************
                                 ;*                                       *
                                 ;*              CONV                     *
                                 ;*                                       *
                                 ;*****************************************

 03EB                            CONV    PROC              NEAR

 03EB  24 0F                             AND     AL, 0FH
 03ED  04 90                             ADD     AL, 090H
 03EF  27                                DAA
 03F0  14 40                             ADC     AL, 040H
 03F2  27                                DAA
```

```
03F3  C3                     RET
                      CONV   ENDP
                      ;*****************************************
                      ;*                                       *
                      ;*      BCD TO ASCII CONVERSION          *
                      ;*                                       *
                      ;*****************************************

= 0030                ASC_BIAS        EQU     030H

03F4                  BCD_ASC         PROC            NEAR

03F4  8A C4                  MOV      AL, AH
03F6  D0 C8                  ROR      AL, 1
03F8  D0 C8                  ROR      AL, 1
03FA  D0 C8                  ROR      AL, 1
03FC  D0 C8                  ROR      AL, 1
03FE  24 0F                  AND      AL, 0FH
0400  04 30                  ADD      AL, ASC_BIAS
0402  88 07                  MOV      [BX], AL
0404  43                     INC      BX
0405  8A C4                  MOV      AL, AH
0407  24 0F                  AND      AL, 0FH
0409  04 30                  ADD      AL, ASC_BIAS
040B  88 07                  MOV      [BX], AL
040D  43                     INC      BX
040E  C6 07 3A               MOV      BYTE PTR [BX], ':'
0411  43                     INC      BX
0412  C3                     RET

                      BCD_ASC         ENDP
                      ;*****************************************
                      ;*                                       *
                      ;*      GET TIME IN MINS AND SECS        *
                      ;*                                       *
                      ;*****************************************

0413                  GET_MS          PROC            NEAR

0413  51                     PUSH     CX
0414  E4 C3           MIN:   IN       AL, MIN_CTR
0416  8A E0                  MOV      AH, AL
0418  E4 D4                  IN       AL, STAT
041A  3C 01                  CMP      AL, 01
041C  74 F6                  JE       MIN
041E  8A EC                  MOV      CH, AH

0420  E4 C2           SEC:   IN       AL, SEC_CTR
0422  8A E0                  MOV      AH, AL
```

```
0424  E4 D4                             IN      AL, STAT
0426  3C 01                             CMP     AL, 01
0428  74 F6                             JE      SEC
042A  8A C5                             MOV     AL, CH

                                  ; mins in AL and secs in AH

042C  59                                POP     CX
042D  C3                                RET

                                  GET_MS          ENDP

                                  ;****************************************
                                  ;*                                      *
                                  ;*      GET TIME IN HOURS               *
                                  ;*                                      *
                                  ;****************************************

042E                              GET_HR          PROC            NEAR

042E  E4 C4                       HR:   IN      AL, HR_CTR
0430  8A E0                             MOV     AH, AL
0432  E4 D4                             IN      AL, STAT
0434  3C 01                             CMP     AL, 01
0436  74 F6                             JE      HR
0438  8A C4                             MOV     AL, AH

                                  ; Hours in AL

043A  C3                                RET

                                  GET_HR          ENDP

                                  ;****************************************
                                  ;*                                      *
                                  ;*   GET DISTANCE COUNT SINCE LAST LOG  *
                                  ;*                                      *
                                  ;****************************************

043B                              GET_DIST        PROC            NEAR

                                  ; get the key to the distance byte
                                  ; get the key to this counter if no other
                                  ; task is currently using it

043B  BB 0020 R                         MOV     BX, OFFSET DIST_LOCK
043E  B8 ---- R                         MOV     AX, SEG DIST_LOCK
0441  8E C0                             MOV     ES, AX
0443  2B C9                             SUB     CX, CX
0445  2B D2                             SUB     DX, DX
0447  B8 0009                           MOV     AX, SC_PEND
044A  CD 20                             INT     VRTX
```

```
044C  BB 022B R                        MOV     BX, OFFSET INC_DIST
044F  8A 07                            MOV     AL, [BX]

                                ; return the key after using resource

0451  BB 0020 R                        MOV     BX, OFFSET DIST_LOCK
0454  B8 ---- R                        MOV     AX, SEG DIST_LOCK
0457  8E C0                            MOV     ES, AX
0459  B9 0001                          MOV     CX, 1
045C  2B D2                            SUB     DX, DX
045E  B8 0008                          MOV     AX, SC_POST
0461  CD 20                            INT     VRTX

0463  C3                               RET

                                GET_DIST        ENDP

                                ;****************************************
                                ;*                                      *
                                ;*  Procedure 'GETS": get a string from *
                                ;*  crt, return pointer in BX           *
                                ;*                                      *
                                ;*  This procedure has been reproduced  *
                                ;*  from the example program provided   *
                                ;*  in Board Support Manual [9] from    *
                                ;*  Hunter and Ready                    *
                                ;*                                      *
                                ;****************************************

0464                            GETS    PROC    NEAR

                                ; Initialize index registers

0464  2B FF                            SUB     DI,DI
0466  B8 000D                   LGETS: MOV     AX, SC_GETC
0469  CD 20                            INT     VRTX

                                ; Echo the character just received

046B  B8 000E                          MOV     AX, SC_PUTC
046E  CD 20                            INT     VRTX

                                ; Save character in buffer & advance index

0470  88 29                            MOV     BYTE PTR [BX][DI], CH
0472  47                               INC     DI

                                ; check for end of line ? Loop back if not

0473  80 FD 0D                         CMP     CH, CR
0476  75 EE                            JNE     LGETS

                                ; If end of line then echo CR and LF too.
                                ; Also put LF in the buffer

 0478  B5 0A                            MOV     CH, LF
 047A  88 29                            MOV     BYTE PTR [BX][DI], CH
 047C  47                               INC     DI
 047D  B8 000E                          MOV     AX, SC_PUTC
 0480  CD 20                            INT     VRTX

                                ; Store end of string marker

 0482  C6 01 00                         MOV     BYTE PTR [BX][DI], EOS
 0485  C3                               RET

                                GETS    ENDP

                                ;***************************************
                                ;*                                     *
                                ;*  Procedure 'PUTS": print a string   *
                                ;*  pointed to by BX                   *
                                ;*                                     *
                                ;*  This procedure has been reproduced *
                                ;*` from the example program provided  *
                                ;*  in Board Support Manual [9] from   *
                                ;*  Hunter and Ready                   *
                                ;*                                     *
                                ;***************************************

 0486                           PUTS    PROC    NEAR

                                ; Initialize index register.
                                ; Get a character to print

 0486  2B F6                            SUB     SI,SI
 0488  8A 28                    LPUTS:  MOV     CH, BYTE PTR [BX][SI]

                                ; Check for end of string - Exit if yes

 048A  80 FD 00                         CMP     CH, EOS
 048D  74 08                            JE      XPUTS

                                ; Echo it if not end of string

 048F  B8 000E                          MOV     AX, SC_PUTC
 0492  CD 20                            INT     VRTX
 0494  46                               INC     SI

                                ; Loop for next character

 0495  EB F1                            JMP     LPUTS

 0497  C3                       XPUTS:  RET
```

```
                                   PUTS    ENDP

                                   ;****************************************
                                   ;*                                      *
                                   ;*  Procedure 'LOG WRITE' writes a five *
                                   ;*  byte log in log table.              *
                                   ;*                                      *
                                   ;****************************************

 0498                              LOG_WRITE      PROC    NEAR          ; entry point

                                   ; assume ds = test data
                                   ; get offset and segment of Log Table

 0498  BE 01EC R                           MOV     SI, OFFSET LOG_BUF
 049B  B8 ---- R                           MOV     AX, LOGS
 049E  8E C0                               MOV     ES, AX

                                   ; get the pointer to next available location

 04A0  BB 00FF R                           MOV     BX, OFFSET ES:LOG_PTR
 04A3  26: 8B 3F                           MOV     DI, ES:[BX]
 04A6  FC                                  CLD
 04A7  B9 0005                             MOV     CX, 05H
 04AA  F3/ A4                      REP     MOVSB
 04AC  26: 89 3F                           MOV     ES:[BX], DI

                                   ; clear accumulated counts in distance
                                   ; byte to start new incremental distance

 04AF  BB 001C R                           MOV     BX, OFFSET NEW_LOG
 04B2  B8 ---- R                           MOV     AX, SEG NEW_LOG
 04B5  8E C0                               MOV     ES, AX
 04B7  B9 0001                             MOV     CX, 1
 04BA  2B D2                               SUB     DX, DX
 04BC  B8 0008                             MOV     AX, SC_POST
 04BF  CD 20                               INT     VRTX

 04C1  C3                                  RET

                                   LOG_WRITE       ENDP

 04C2                              TEST_CODE       ENDS
```

```
                                  page
                                  ;****************************************
                                  ;*                                      *
                                  ;*           Data for program           *
                                  ;*    mailboxes, messages and buffer    *
                                  ;*                                      *
                                  ;****************************************
0000                              TEST_DATA     SEGMENT  PARA    PUBLIC  'DATA'
                                  ; mailbox for locking crt
0000  01 00 00 00                 CRT     DD      1
                                  ; data buffer for crt input
0004  0100[                       BUF     DB      256 DUP (EOS)
          00
                         ]

                                  ; newline sequence <cr lf>
0104  0D 0A 00                    NL      DB      CR, LF, EOS
                                  ; Messages
0107  56 52 54 58 20 2D 20        MSG1    DB      'VRTX - I/O test started'
      49 2F 4F 20 74 65 73
      74 20 73 74 61 72 74
      65 64
011E  0D 0A 00                            DB      CR, LF, EOS
0121  4D 75 6C 74 69 74 61        MSG2    DB      'Multitasking using VRTX '
      73 6B 69 6E 67 20 75
      73 69 6E 67 20 56 52
      54 58 20
0139  0D 0A 00                            DB      CR, LF, EOS
013C  0D 0A                       MSG3    DB      CR, LF
013E  63 6F 6E 74 72 6F 6C                DB      'control C received,'
      20 43 20 72 65 63 65
      69 76 65 64 2C
0151  20 70 6C 65 61 73 65                DB      ' please type in a line'
      20 74 79 70 65 20 69
      6E 20 61 20 6C 69 6E
      65
0167  0D 0A 00                            DB      CR, LF, EOS
016A  3D 3E 00                    MSG4    DB      '=>', EOS
016D  0D 0A                       MSG5    DB      CR, LF
016F  50 6F 77 65 72 20 4F                DB      'Power On Log Created'
```

```
       6E 20 4C 6F 67 20 43
       72 65 61 74 65 64
0183   0D 0A 00                              DB       CR, LF, EOS

0186   0D 0A                       MSG7      DB       CR, LF
0188   4F 6E 65 20 4D 69 6E                  DB       'One Minute Idle Log Created'
       75 74 65 20 49 64 6C
       65 20 4C 6F 67 20 43
       72 65 61 74 65 64
01A3   0D 0A 0A 00                           DB        CR, LF, LF, EOS

01A7   0D 0A                       MSG8      DB       CR, LF
01A9   54 77 6F 20 4D 69 6E                  DB       'Two Minute Idle Log Created'
       75 74 65 20 49 64 6C
       65 20 4C 6F 67 20 43
       72 65 61 74 65 64
01C4   0D 0A 0A 00                           DB        CR, LF, LF, EOS

01C8   0D 0A                       MSG10     DB       CR, LF
01CA   45 78 74 65 72 6E 61                  DB       'External Input Log Created'
       6C 20 49 6E 70 75 74
       20 4C 6F 67 20 43 72
       65 61 74 65 64
01E4   0D 0A 0A 00                           DB       CR, LF, LF, EOS

01E8   00 00 00 00                 HALT_TIME          DD       0

                                   ; buffers for temporary storage of logs, time e
                                   tc

01EC   0005[                       LOG_BUF            DB       5  DUP (?)
            ??
                   ]

01F1   0012[                       DSP_BUF            DB       18 DUP (0)
            00
                   ]

0203   0010[                       TIME               DB       16 DUP (?)
            ??
                   ]

0213   0010[                       REAL_TIME          DB       16 DUP (?)
            ??
                   ]

                                   ; Sensor beam related temp store

                                             PUBLIC   LAST_WD
0223   0000                        LAST_WD            DW       0
                                             PUBLIC   B_STAT
0225   0000                        B_STAT             DW       0
```

```
                                ; Distance related temp store

                                        PUBLIC  DIST
0227  00 00 00 00               DIST            DD      0
022B  F0                        INC_DIST        DB      240

022C  0D 0A                     RT_MSG  DB      CR, LF
022E  52 65 61 6C 20 54 69              DB      'Real Time (hh:mm:ss) is = '
      6D 65 20 28 68 68 3A
      6D 6D 3A 73 73 29 20
      69 73 20 3D 20
0248  00                                DB      EOS

0249                            TEST_DATA       ENDS

0000                            LOGS    SEGMENT         PARA    'DATA'

                                ; All logs are stored here and log ptr
                                ; points to next empty location where
                                ; a new log can be stored

0000  00FF[                             LOG_TBL         DB      255 DUP(0)
          00
                       ]

00FF  0000                              LOG_PTR         DW      0

0101                            LOGS    ENDS

0000                            MBOX    SEGMENT PARA    PUBLIC 'DATA'

                                ; All mailboxes are defined here

                                ; odometer mailboxes

                                        PUBLIC  MBOX_OD1
0000  00 00 00 00                       MBOX_OD1        DD      0
                                        PUBLIC  MBOX_OD2
0004  00 00 00 00                       MBOX_OD2        DD      0
                                        PUBLIC  MBOX_OD3
0008  00 00 00 00                       MBOX_OD3        DD      0
                                        PUBLIC  MBOX_OD4
000C  00 00 00 00                       MBOX_OD4        DD      0
                                        PUBLIC  MBOX_OD5
0010  00 00 00 00                       MBOX_OD5        DD      0

                                ; mailboxes for log writting

                                        PUBLIC  LOG_1MIN
0014  00 00 00 00                       LOG_1MIN        DD      0
```

```
0018  00 00 00 00                       LOG_2MIN        DD      0
001C  00 00 00 00                       NEW_LOG         DD      0

                                ; mailboxes for distance maintenance

0020  01 00 00 00                       DIST_LOCK       DD      1
0024  00 00 00 00                       DIST_OF         DD      0

                                ; mailbox for signpost

                                        PUBLIC  SIGN_BOX
0028  00 00 00 00                       SIGN_BOX        DD      0

                                ; mailboxes for sensor interrupts

                                        PUBLIC  MBOX_E1
002C  00 00 00 00                       MBOX_E1         DD      0
                                        PUBLIC  MBOX_E2
0030  00 00 00 00                       MBOX_E2         DD      0
                                        PUBLIC  MBOX_E3
0034  00 00 00 00                       MBOX_E3         DD      0
                                        PUBLIC  MBOX_E4
0038  00 00 00 00                       MBOX_E4         DD      0
                                        PUBLIC  MBOX_E5
003C  00 00 00 00                       MBOX_E5         DD      0
                                        PUBLIC  MBOX_E6
0040  00 00 00 00                       MBOX_E6         DD      0
                                        PUBLIC  MBOX_E7
0044  00 00 00 00                       MBOX_E7         DD      0
                                        PUBLIC  MBOX_E8
0048  00 00 00 00                       MBOX_E8         DD      0
                                        PUBLIC  MBOX_E9
004C  00 00 00 00                       MBOX_E9         DD      0
                                        PUBLIC  MBOX_E10
0050  00 00 00 00                       MBOX_E10        DD      0
                                        PUBLIC  MBOX_E11
0054  00 00 00 00                       MBOX_E11        DD      0
                                        PUBLIC  MBOX_E12
0058  00 00 00 00                       MBOX_E12        DD      0
                                        PUBLIC  MBOX_E13
005C  00 00 00 00                       MBOX_E13        DD      0
                                        PUBLIC  MBOX_E14
0060  00 00 00 00                       MBOX_E14        DD      0
                                        PUBLIC  MBOX_E15
0064  00 00 00 00                       MBOX_E15        DD      0
                                        PUBLIC  MBOX_E16
0068  00 00 00 00                       MBOX_E16        DD      0

006C                            MBOX    ENDS

                                        END
```

```
Segments and Groups:

                N a m e                     Size    Align   Combine Class

CFTBL . . . . . . . . . . . . . . . .       002C    PARA    NONE    'DATA'
LOGS . . . . . . . . . . . . . . . .        0101    PARA    NONE    'DATA'
MBOX . . . . . . . . . . . . . . . .        006C    PARA    PUBLIC  'DATA'
TEST_CODE . . . . . . . . . . . . . .       04C2    PARA    NONE    'CODE'
TEST_DATA . . . . . . . . . . . . . .       0249    PARA    PUBLIC  'DATA'

Symbols:

                N a m e                     Type    Value   Attr

ASC  . . . . . . . . . . . . . . . .        N PROC  039B    TEST_CODE       Length = 001D
ASC_BIAS . . . . . . . . . . . . . .        Number  0030

BCD_ASC  . . . . . . . . . . . . . .        N PROC  03F4    TEST_CODE       Length = 001F
BUF  . . . . . . . . . . . . . . . .        L BYTE  0004    TEST_DATA       Length = 0100
B_STAT . . . . . . . . . . . . . . .        L WORD  0225    TEST_DATA       Global

CONV . . . . . . . . . . . . . . . .        N PROC  03EB    TEST_CODE       Length = 0009
CR . . . . . . . . . . . . . . . . .        Number  000D
CRT  . . . . . . . . . . . . . . . .        L DWORD 0000    TEST_DATA
CTLC . . . . . . . . . . . . . . . .        N PROC  012B    TEST_CODE       Length = 0053

DIST . . . . . . . . . . . . . . . .        L DWORD 0227    TEST_DATA       Global
DIST_LOCK  . . . . . . . . . . . . .        L DWORD 0020    MBOX
DIST_OF  . . . . . . . . . . . . . .        L DWORD 0024    MBOX
DSP_BUF  . . . . . . . . . . . . . .        L BYTE  01F1    TEST_DATA       Length = 0012

EOS  . . . . . . . . . . . . . . . .        Number  0000
ER_BUF . . . . . . . . . . . . . . .        Number  0007
ER_COM . . . . . . . . . . . . . . .        Number  0021
ER_CVT . . . . . . . . . . . . . . .        Number  0020
ER_INI . . . . . . . . . . . . . . .        Number  000F
ER_ISC . . . . . . . . . . . . . . .        Number  0009
ER_MEM . . . . . . . . . . . . . . .        Number  0003
ER_MIU . . . . . . . . . . . . . . .        Number  0005
ER_NCP . . . . . . . . . . . . . . .        Number  0010
ER_NMB . . . . . . . . . . . . . . .        Number  0004
ER_NMP . . . . . . . . . . . . . . .        Number  000B
ER_OPC . . . . . . . . . . . . . . .        Number  0022
ER_PID . . . . . . . . . . . . . . .        Number  000E
ER_QFL . . . . . . . . . . . . . . .        Number  000D
ER_QID . . . . . . . . . . . . . . .        Number  000C
ER_TCB . . . . . . . . . . . . . . .        Number  0002
ER_TID . . . . . . . . . . . . . . .        Number  0001
ER_TMO . . . . . . . . . . . . . . .        Number  000A
ER_WIC . . . . . . . . . . . . . . .        Number  0008
ER_ZMW . . . . . . . . . . . . . . .        Number  0006
EXT_IP . . . . . . . . . . . . . . .        N PROC  033F    TEST_CODE       Length = 005C

GETS . . . . . . . . . . . . . . . .        N PROC  0464    TEST_CODE       Length = 0022
```

```
GET_DIST . . . . . . . . . . . . . .    N PROC   043B   TEST_CODE       Length = 0029
GET_HR . . . . . . . . . . . . . . .    N PROC   042E   TEST_CODE       Length = 000D
GET_MS . . . . . . . . . . . . . . .    N PROC   0413   TEST_CODE       Length = 001B
GO_RTC . . . . . . . . . . . . . . .    Number   00D5

HALT_TIME  . . . . . . . . . . . . .    L DWORD  01E8   TEST_DATA
HR . . . . . . . . . . . . . . . . .    L NEAR   042E   TEST_CODE
HR_CTR . . . . . . . . . . . . . . .    Number   00C4

IDL  . . . . . . . . . . . . . . . .    L NEAR   0298   TEST_CODE
IDL2 . . . . . . . . . . . . . . . .    L NEAR   02E8   TEST_CODE
IDLE1  . . . . . . . . . . . . . . .    N PROC   0298   TEST_CODE       Length = 0050
IDLE2  . . . . . . . . . . . . . . .    N PROC   02E8   TEST_CODE       Length = 0057
IDLE_1M  . . . . . . . . . . . . . .    N PROC   01DF   TEST_CODE       Length = 004B
IDLE_2M  . . . . . . . . . . . . . .    N PROC   022A   TEST_CODE       Length = 006E
INC_DIST . . . . . . . . . . . . . .    L BYTE   022B   TEST_DATA

LAST_WD  . . . . . . . . . . . . . .    L WORD   0223   TEST_DATA       Global
LF . . . . . . . . . . . . . . . . .    Number   000A
LGETS  . . . . . . . . . . . . . . .    L NEAR   0466   TEST_CODE
LOG1 . . . . . . . . . . . . . . . .    Number   0001
LOG10  . . . . . . . . . . . . . . .    Number   000A
LOG11  . . . . . . . . . . . . . . .    Number   000B
LOG12  . . . . . . . . . . . . . . .    Number   000C
LOG2 . . . . . . . . . . . . . . . .    Number   0002
LOG3 . . . . . . . . . . . . . . . .    Number   0003
LOG4 . . . . . . . . . . . . . . . .    Number   0004
LOG5 . . . . . . . . . . . . . . . .    Number   0005
LOG6 . . . . . . . . . . . . . . . .    Number   0006
LOG7 . . . . . . . . . . . . . . . .    Number   0007
LOG8 . . . . . . . . . . . . . . . .    Number   0008
LOG9 . . . . . . . . . . . . . . . .    Number   0009
LOG_1MIN . . . . . . . . . . . . . .    L DWORD  0014   MBOX    Global
LOG_2MIN . . . . . . . . . . . . . .    L DWORD  0018   MBOX
LOG_BUF  . . . . . . . . . . . . . .    L BYTE   01EC   TEST_DATA       Length = 0005
LOG_PTR  . . . . . . . . . . . . . .    L WORD   00FF   LOGS
LOG_TBL  . . . . . . . . . . . . . .    L BYTE   0000   LOGS    Length = 00FF
LOG_WRITE  . . . . . . . . . . . . .    N PROC   0498   TEST_CODE       Length = 002A
LPUTS  . . . . . . . . . . . . . . .    L NEAR   0488   TEST_CODE

M1 . . . . . . . . . . . . . . . . .    L NEAR   01DF   TEST_CODE
M10  . . . . . . . . . . . . . . . .    L NEAR   0205   TEST_CODE
M2 . . . . . . . . . . . . . . . . .    L NEAR   022A   TEST_CODE
M20  . . . . . . . . . . . . . . . .    L NEAR   0257   TEST_CODE
MAIN . . . . . . . . . . . . . . . .    F PROC   0000   TEST_CODE       Global  Length = 00B7
MBOX_E1  . . . . . . . . . . . . . .    L DWORD  002C   MBOX    Global
MBOX_E10 . . . . . . . . . . . . . .    L DWORD  0050   MBOX    Global
MBOX_E11 . . . . . . . . . . . . . .    L DWORD  0054   MBOX    Global
MBOX_E12 . . . . . . . . . . . . . .    L DWORD  0058   MBOX    Global
MBOX_E13 . . . . . . . . . . . . . .    L DWORD  005C   MBOX    Global
MBOX_E14 . . . . . . . . . . . . . .    L DWORD  0060   MBOX    Global
MBOX_E15 . . . . . . . . . . . . . .    L DWORD  0064   MBOX    Global
MBOX_E16 . . . . . . . . . . . . . .    L DWORD  0068   MBOX    Global
MBOX_E2  . . . . . . . . . . . . . .    L DWORD  0030   MBOX    Global
```

```
MBOX_E3 . . . . . . . . . . . . . .     L DWORD 0034    MBOX      Global
MBOX_E4 . . . . . . . . . . . . . .     L DWORD 0038    MBOX      Global
MBOX_E5 . . . . . . . . . . . . . .     L DWORD 003C    MBOX      Global
MBOX_E6 . . . . . . . . . . . . . .     L DWORD 0040    MBOX      Global
MBOX_E7 . . . . . . . . . . . . . .     L DWORD 0044    MBOX      Global
MBOX_E8 . . . . . . . . . . . . . .     L DWORD 0048    MBOX      Global
MBOX_E9 . . . . . . . . . . . . . .     L DWORD 004C    MBOX      Global
MBOX_OD1 . . . . . . . . . . . . . .    L DWORD 0000    MBOX      Global
MBOX_OD2 . . . . . . . . . . . . . .    L DWORD 0004    MBOX      Global
MBOX_OD3 . . . . . . . . . . . . . .    L DWORD 0008    MBOX      Global
MBOX_OD4 . . . . . . . . . . . . . .    L DWORD 000C    MBOX      Global
MBOX_OD5 . . . . . . . . . . . . . .    L DWORD 0010    MBOX      Global
MESSAGE . . . . . . . . . . . . . .     N PROC  00F8    TEST_CODE         Length = 0033
MIN  . . . . . . . . . . . . . . . .    L NEAR  0414    TEST_CODE
MIN_CTR . . . . . . . . . . . . . .     Number  00C3
MSGI . . . . . . . . . . . . . . . .    L BYTE  0107    TEST_DATA
MSG10 . . . . . . . . . . . . . . .     L BYTE  01C8    TEST_DATA
MSG2 . . . . . . . . . . . . . . . .    L BYTE  0121    TEST_DATA
MSG3 . . . . . . . . . . . . . . . .    L BYTE  013C    TEST_DATA
MSG4 . . . . . . . . . . . . . . . .    L BYTE  016A    TEST_DATA
MSG5 . . . . . . . . . . . . . . . .    L BYTE  016D    TEST_DATA
MSG7 . . . . . . . . . . . . . . . .    L BYTE  0186    TEST_DATA
MSG8 . . . . . . . . . . . . . . . .    L BYTE  01A7    TEST_DATA

NEW_LOG . . . . . . . . . . . . . .     L DWORD 001C    MBOX
NL . . . . . . . . . . . . . . . . .    L BYTE  0104    TEST_DATA

ONE_MIN . . . . . . . . . . . . . .     Number  1770
OPN . . . . . . . . . . . . . . . .     N PROC  03B8    TEST_CODE         Length = 0033
OVER . . . . . . . . . . . . . . . .    L NEAR  03AC    TEST_CODE

PRMSG . . . . . . . . . . . . . . .     L NEAR  00F8    TEST_CODE
PUTS . . . . . . . . . . . . . . . .    N PROC  0486    TEST_CODE         Length = 0012
PWR_ON . . . . . . . . . . . . . . .    N PROC  00B7    TEST_CODE         Length = 0041

REAL_TIME . . . . . . . . . . . . .     L BYTE  0213    TEST_DATA         Length = 0010
RET_OK . . . . . . . . . . . . . . .    Number  0000
RT . . . . . . . . . . . . . . . . .    L NEAR  017E    TEST_CODE
RTASK . . . . . . . . . . . . . . .     N PROC  017E    TEST_CODE         Length = 0061
RT_MSG . . . . . . . . . . . . . . .    L BYTE  022C    TEST_DATA

SC_ACCEPT . . . . . . . . . . . . .     Number  0025
SC_GBLOCK . . . . . . . . . . . . .     Number  0006
SC_GETC . . . . . . . . . . . . . .     Number  000D
SC_GTIME . . . . . . . . . . . . . .    Number  000A
SC_LOCK . . . . . . . . . . . . . .     Number  0020
SC_PCREATE . . . . . . . . . . . . .    Number  0022
SC_PEND . . . . . . . . . . . . . .     Number  0009
SC_PEXTEND . . . . . . . . . . . . .    Number  0023
SC_POST . . . . . . . . . . . . . .     Number  0008
SC_PUTC . . . . . . . . . . . . . .     Number  000E
SC_QACCEPT . . . . . . . . . . . . .    Number  0028
SC_QCREATE . . . . . . . . . . . . .    Number  0029
SC_QINQUIRY . . . . . . . . . . . .     Number  002A
```

```
SC_QPEND . . . . . . . . . . . .        Number   0027
SC_QPOST . . . . . . . . . . . .        Number   0026
SC_RBLOCK  . . . . . . . . . . .        Number   0007
SC_STIME . . . . . . . . . . . .        Number   000B
SC_TCREATE . . . . . . . . . . .        Number   0000
SC_TDELAY  . . . . . . . . . . .        Number   000C
SC_TDELETE . . . . . . . . . . .        Number   0001
SC_TINQUIRY  . . . . . . . . . .        Number   0005
SC_TPRIORITY . . . . . . . . . .        Number   0004
SC_TRESUME . . . . . . . . . . .        Number   0003
SC_TSLICE  . . . . . . . . . . .        Number   0015
SC_TSUSPEND  . . . . . . . . . .        Number   0002
SC_UNLOCK  . . . . . . . . . . .        Number   0021
SC_WAITC . . . . . . . . . . . .        Number   000F
SEC  . . . . . . . . . . . . . .        L NEAR   0420    TEST_CODE
SEC_CIR  . . . . . . . . . . . .        Number   00C2
SGNP . . . . . . . . . . . . . .        L NEAR   033F    TEST_CODE
SIGN_BOX . . . . . . . . . . . .        L DWORD  0028    MBOX     Global
STAT . . . . . . . . . . . . . .        Number   00D4

TBL  . . . . . . . . . . . . . .        L BYTE   0000    CFTBL    Global
TIME . . . . . . . . . . . . . .        L BYTE   0203    TEST_DATA        Length = 0010
TWO_MIN  . . . . . . . . . . . .        Number   2EE0
TXRDY  . . . . . . . . . . . . .        L FAR    0000             External

UI_ENTER . . . . . . . . . . . .        Number   0016
UI_EXIT  . . . . . . . . . . . .        Number   0011
UI_RXCHR . . . . . . . . . . . .        Number   0013
UI_TIMER . . . . . . . . . . . .        Number   0012
UI_TXRDY . . . . . . . . . . . .        Number   0014

VRTX . . . . . . . . . . . . . .        Number   0020
VRTX_GO  . . . . . . . . . . . .        Number   0031
VRTX_INIT  . . . . . . . . . . .        Number   0030

WAITC  . . . . . . . . . . . . .        L NEAR   012B    TEST_CODE

XPUTS  . . . . . . . . . . . . .        L NEAR   0497    TEST_CODE

ZERO . . . . . . . . . . . . . .        L NEAR   00C0    TEST_CODE

   1464 Source  Lines
   1468 Total   Lines
    196 Symbols

  44114 Bytes symbol space free

      0 Warning Errors
      0 Severe  Errors
```